IN THE UNITED STATES PATENT & TRADEMARK OFFICE

In re App. No.:	09/449,021	)	PATENT APPLICATION
		)	
Filing Date:	November 24, 1999	)	Art Unit: 2192
		)	
Inventors:	Emmelmann	)	Examiner: C. Kendall
		)	
Title:	*Interactive Server Side Components*	)	
		)	Customer No.: 28554

APPELLANT'S BRIEF

APPELLANT'S BRIEF

TABLE OF CONTENTS

APPELLANT'S BRIEF

I. REAL PARTY IN INTEREST

The sole inventor is applicant and the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF THE CLAIMS

Claims 1-6, 8, 22-33, 41-43, 51-96 and 114-128 stand finally rejected in the Office Action dated January 21, 2010, and are the subject of this appeal.

Claims 9-21, 34-40, 44-50 and 97-113 have been previously withdrawn as drawn to non-elected subject matter. Claim 7 has been cancelled.

Applicant respectfully notes that this application has been pending since November 24, 1999, with priority dating to November 25 1998, and was made Special by the Decision on Petition grated June 30, 2009. Further, a Notice of Allowance was issued for all claims pending in this Appeal on April 15, 2008. The PTO subsequently withdrew the Notice of Allowance on July 9, 2008, apparently due to a Quality Review, and the Office Action which followed on September 17, 2008 rejected all pending claims on the basis of prior art that Applicant had cited to the PTO on March 6, 2000, and which was initialed as considered by the Examiner on March 21, 2001. This same prior art (WebWriter I) remains the main issue for this appeal. It is combined with a later article (WebWriter II) by the same company, which is mainly relevant for the dependent claims, and which is significantly different in several key respects, as discussed below.

IV. STATUS OF AMENDMENTS

There were no amendments proposed after the final rejection, and Appendix A is the current list of pending claims.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Applicant has developed a unique editor for editing server-based ***dynamic web applications***. A server-based ***dynamic web application*** is a software program that generates web pages upon request by a browser. In general, a user visiting a web site that is running a ***dynamic web application*** receives different versions of the same web page when viewing it multiple times. (page 1:16-18; Fig. 6). This makes it difficult to edit the application, because the end result does not always look the same.

Applicant describes an embodiment that uses "page templates" and "components" to build server-based ***dynamic web applications*** (page 8:1-25). Instead of programming a web application from scratch, the idea is to create the application by connecting preexisting software components (page 4:16-18 and page 5:7-12). Applicant's components are program modules or program classes containing instructions to generate browser code, possibly instructions to react on user input (pages 8:28 - 9:7; pages 19:26-20:12) and instructions to assist in editing. Applicant's page templates are used to specify how the components are to be connected to each other and to the layout (page 5:7-9 and page 8:1- 4). When the user visits the web application, page templates (26) and especially components (27) are executed (Fig. 7; page 8:28; page 16:14-18 (execution of components); and page 46:3-23 (execution of templates)) and thus transformed into the generated pages and sent to the browser (23) (page 15:3-5) for display to the user.

Providing true WYSIWYG ("What You See Is What You Get") functionality for editing of dynamic web applications is a complicated problem, because pages dynamically change during execution of the web application. Thus, using a conventional editor, there is no single page view that could be shown to a developer for editing. Applicant addresses this problem by running the web application in the editor (pages 3:27-4:2).

Advantageously, in applicant's editor (*see* Section C on page 25 to 40), the application looks similar to and functions like the normal running application as viewed by the end user, with the addition of editing features such as handles. (*See* page 4:29-30; page 5:20-21; Fig. 1 (sample application as viewed by the end user); Fig. 2 (view in applicant's editor); and page 27:12 (handles)). Conventional editing technology (such as the cited prior art) does not really address this issue; in general, page templates look different when viewed during editing than when the generated page is viewed by the user (pages 3:29-4:4). In contrast, applicant describes

a special page generator (162) that runs the application generating pages with editing features so that the editor shows the generated pages (163) to the developer (pages 4:29-5-4; pages 5:21-24; page 26:5-7; and Fig. 18) while actually applying changes to the page template (161) (page 5:25-26; page 26:10-13, for page generator, see pages 26:26-27:2, and Fig. 18; for editing features, see element 164 and section 5b on page 30, and section 5d on page 31). Components cooperate in editing (Fig. 11 dotted parts, Fig 12, section 6 on page 18, section 5d on page 31) e.g. by correctly positioning handles. In addition, the editor comprises a client part in form of scripts for download into the browser (Fig. 18, element 165; section 6 on page 32-36) to process user interactions and a server part for changing the page templates (element 166, section 7 on pages 36-40). Using applicant's editor would cause the display of an application, for example, a shopping application, to have a functional shopping basket thereby allowing the developer to add and/or remove items from the basket and to see the end user's actual view during editing.

Advantageously, Applicant's components operate on the server yet can still interact with the user via multiple page requests (page 4:21-22; page 5:13-19; page 9:28 to page 10:4; pages 11:21-13:30). Thereby, the number and kind of components per generated page can change dynamically (page 4:23-28; page 11: 7-19). Components are denoted using tag syntax and can be nested (section 1 on page 8). The preferred implementation of applicant's interactive server side components (ISSC) is described in section B on pages 14 to 24 and in Section D on pages 40 to 51, which discloses how the object oriented language heitml is used to implement page templates and ISSC's.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 1-6, 8, 22-33, 41-43, 51-96 and 114-128 are unpatentable under Section 103(a) over the combination of the 1996 article by Crespo and Bier entitled: *WebWriter: A Browser-Based Editor for Constructing Web Applications* ("WebWriter I") and the 1997 article by Crespo, Chang and Bier entitled *Responsive Interaction for a Large Web Application: The Meteor Shower Architecture in the WebWriter II Editor* ("WebWriter II").

VII. ARGUMENTS

ALL CLAIMS ARE PATENTABLE OVER THE COMBINATION OF WEBWRITER I AND WEBWRITER II.

INTRODUCTION

A key issue in this appeal involves a substantial disagreement over the principal function of the prior art, and consequently over what is taught by the prior art articles which are cited against the claims. In one important example, Applicant maintains that the cited art fails to teach or suggest the feature of running an application while editing this very same application, a feature that is recited in many of the pending claims. As a result, the examiner's reasoning regarding these claims is flawed because the prior art articles disclose running the application after editing, not during editing, and so there is no information flow from the running application back to the editor. Section A below addresses this critical disagreement in detail. Further, the Examiner appears to rely solely upon this flawed reasoning as applicable to all claims in the latest Office Action, and thus, other distinctive features of the claims are not addressed.

If the Board agrees with Applicant and concludes that the WebWriter articles fail to teach or suggest this critical feature, then applicant submits that all claims are allowable as pending, and only the discussion of the independent claims in section B below needs to be reviewed. If not, then there are additional features recited in various independent claims as discussed in section B, and also in various dependent claims as discussed in Section D below. Also, many of the dependent claims contain more detailed claim language with regard to certain distinctive features recited in the independent claims. Grouping of claims is discussed where appropriate in Sections B and D.

A. THE EXAMINER IS MISTAKEN REGARDING THE NATURE AND SCOPE OF THE CITED PRIOR ART

i. Overview

The Examiner asserts that the combination of the WebWriter I article and the WebWriter II article makes Applicant's claims obvious. However, Applicant respectfully disagrees, and continues to believe that the Patent Office is mistaken regarding the nature and scope of the disclosure in the WebWriter articles.

Classically, software applications are created and/or edited at one time (editing-time) using an editor program, and then run at a different time (run-time). The application is first being edited then run, but not both.

Beneficially, applicant's editor runs the edited application while it is being edited. Applicant continues to assert that neither of the cited articles describes or suggests running edited applications during editing. In contrast, the "Examiner maintains that WebWriter does in fact have this limitation" (*See* Final Office Action at p. 21).

Applicant and examiner both rely on the same citation from WebWriter I:

> "*template page, which includes the specification of locations within the page, called dynamic areas where computed content should be inserted* ***at run-time*** *to produce the final page that will be displayed to the user*" (emphasis added) (*see* WebWriter I at page 2, left column, last paragraph)

The specific use of the phrase "***at run-time***" makes clear that indeed "running" takes place at run-time and not during editing in WebWriter. The examiner has not refuted applicant's argument about the use of term "at run-time" in the cited material, and the examiner's position is simply wrong since the cited portion actually supports applicant's position. Applicant gives more arguments and citations about this issue in sections A.ii, C.i and C.ii.

The WebWriter I article describes a software development system consisting of two distinct programs: the WebWriter Editor and the WebWriter Page Generator. In the final office action, it is interesting that the examiner only refers to the WebWriter Page Generator as prior art for applicant's editor (see page 3), and not the WebWriter Editor. The program named WebWriter Page Generator, however, is a separate program unrelated to editing for use at run-time after editing. The WebWriter Page Generator belongs to the WebWriter development system, but it does not have any features for editing, and it is not used by the WebWriter Editor; instead the WebWriter Editor has its own page generation part (see section C.i for details). The fact that the examiner does not even cite the WebWriter editor against the claims shows that the examiner's reasoning is flawed. The examiner apparently believes that the WebWriter page generator is part of the WebWriter editor or that both programs cooperate. However, that is simply not true, as discussed in section C.i.

The Web Writer editor described in WebWriter I is mainly concerned with editing the static parts of a page, and shows dynamic parts as placeholders during editing (*See* Figure 7b,

reproduced below). For example, the WebWriter I article states that "*static parts of each page are created in advance by the designer using a text editor*" (*see* p. 2, 2nd para. under heading "The WebWriter Application Model") and further, "*WebWriter then adds a placeholder for the output area [which] will be replaced* ***at runtime*** *by the output of a script . . . .*" (*see* page 4, 2nd para. under heading "Placing Dynamic Areas", emphasis added).

In contrast, Applicant's invention is concerned with editing components, which are dynamic parts. Further, Applicant's invention actually displays the executed dynamic parts during editing, and except for the addition of editing features, e.g., handles, the appearance and function of the content during editing is virtually the same as at run-time. This is achieved by running the edited web application during editing.

Some of Applicant's claims directly require the feature of (1) the edited application running during editing. Other claims recite features, such as (2) the component displayed during editing has a similar appearance as on the generated page with the addition of editing features, or (3) components cooperate with the editor during editing. In addition, there are other distinctive features recited in various claims, like (4) a varying component set, (5) nested components (*see* section B.ix), and (6) interactive components (*see* section D.ii).

For example, claims 1 and 59 require a document generator that runs the application during editing, claim 90 requires that scripts in the edited document remain running during editing, and method claim 125 explicitly requires a running step during editing. Claim 26 requires that the documents be similar at run time and during editing, to wit: "*at least a part of the second document appears and functions similar to the first document.*" Claims 74 and 114 require components that cooperate with the editor during editing.

Section A.ii below contains a more detailed discussion explaining why the WebWriter articles do not teach or suggest running applications during editing. Section C then gives a very detailed discussion on certain aspects of the prior art; especially section C.iii, which discusses why the combination of WebWriter I and WebWriter II is improper.

ii. The Examiner Misinterprets WebWriter

As discussed in the overview above, the examiner believes that one or both of the WebWriter articles teach that at least part of the edited application is executed during editing, while applicant believes that the WebWriter articles only describe executing the application after

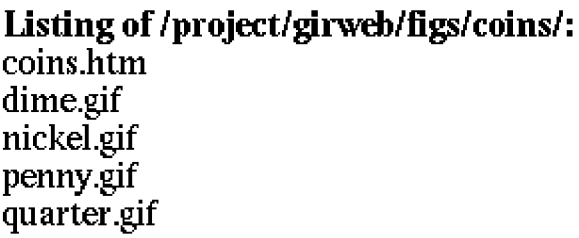
editing, and not during editing. Further support for applicant's position on this issue is found within the WebWriter I article. For example, on p. 4, 2nd paragraph, under the heading "Placing Dynamic Areas," it is stated that "*WebWriter then adds a placeholder for the output area [which] will be replaced* ***at runtime*** *by the output of a script . . . .*" (emphasis added). This text makes clear that WebWriter does have the runtime restriction as the citation discussed in A.i., i.e., the script is replaced by a placeholder, and is not executed except at runtime. Also, on p. 6, col. 2, in the section entitled "Running the Application," the article states that "***Once*** *a Web Writer application is created and* ***saved*** *to disk it can be* ***run*** *in one of two ways*" (emphasis added). This text makes clear that running takes place after saving the edited file, not during the editing process.

In addition, the WebWriter Page Generator that is used in the WebWriter system to execute an application is a separate program not having any features for editing and not being used by the WebWriter Editor. The WebWriter Editor has its own page generator built in for displaying page templates. This becomes clear from reviewing the implementation description of the two programs in the WebWriter article as detailed in section C.i. The fact that these programs are separate programs, and that WebWriter teaches away from using the WebWriter page generator during editing by including another distinct page generator into the editor, makes clear that running of the application does not take place during editing in WebWriter. Finally, the editor architecture of WebWriter does not support running during editing as detailed in section C.ii, e.g., a page template is represented as a tree during editing, but needs to be a text file for execution.

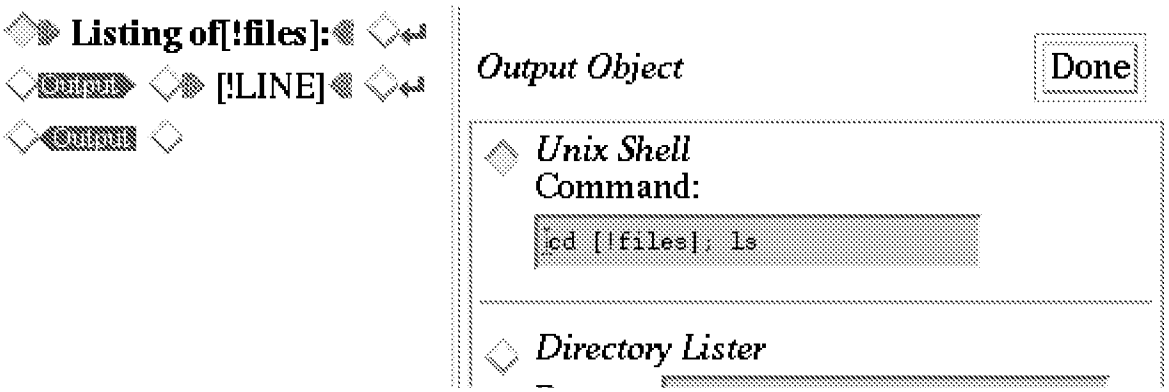
During editing, WebWriter I only shows the static parts of a template page while dynamically generated parts are displayed as static placeholders. Upon completion of editing, the template page is saved. Thereafter, at runtime, the placeholders are replaced with dynamically generated parts, which are then executed and their output is displayed. This is explained as noted in Section A.i above.

As a result, the output generated by scripts at run time generally looks very different than the placeholders displayed while editing, so that pages displayed by WebWriter during editing and at runtime usually look very different as well. For example, as shown below, Fig. 7(b) of the WebWriter I reference shows the editing view and Fig. 6(b) shows the runtime view of a single page template implementing a file listing application. The runtime view correctly shows a list of

files, while the editing view shows a placeholder consisting of handles surrounding the output formatting specification *"[!Line]."*



WebWriter Fig. 6(b) –View while Running



WebWriter Fig. 7(b) –View while Editing

Also, Fig. 10(b) and Fig. 11 show the difference between the runtime and editing views in WebWriter I of a page template implementing a walkie talkie application. The runtime view in Fig. 10(b) shows the text of a message received by the walkie talkie while the editing view just shows the placeholder, namely *"[!Line]"* surrounded by handles.

On page 25 of the Office Action dated July 1, 2009, the examiner argues that merely adding the limitation of having similar appearance is not distinctive. However, applicant notes that WYSIWYG editing was a significant breakthrough in editing technology by showing an editing view that is similar to the final end result shown to the user. Therefore, applicant sees the limitation of having a similar appearance as a very significant limitation rather than a minor

limitation. It is a problem of WebWriter and many other editors of the time that this WYSIWYG principle is not followed for the dynamic portions of the pages.

B. THE INDEPENDENT CLAIMS ARE PATENTABLE OVER THE COMBINATION OF WEBWRITER I AND WEBWRITER II

The claims do not stand or fall together, but should be considered separately patentable as discussed below.

i. Claim 1

a) Grouping

Independent Claim 1 forms the basis for a first group considered to stand or fall together, and not with other claims or groups. Claim 1 is an independent claim that is directed to a basic two-part structure that runs a web application and an editor.

A key distinction from other claims is that Claim 1 does not rely on components at all. Therefore, Claim 1 should be considered separately with respect to all other claims reciting components, namely independent claims 6, 22, 51, 74, 114, 125 and corresponding dependent claims, as well as dependent claims 2, 27, 64 and 94, because an editor editing running applications without components, but just using scripts, for example, would be possible and useful as well. Also, because the listed independent claims require specific important distinctive features, such as executing components during editing, or components cooperating with the editor, which make these claims non-obvious over claim 1; more details regarding individual claims are discussed below.

Claim 1 requires including editing features in generated pages and the editor program operating via the editing features. It was not obvious at the time the invention was made, to include the editing features in the generated pages which resulted from at least partly running the edited application and to make the editor program operate via these editing features. In fact, most editors do not operate via editing features in generated pages. WebWriter inserts some editing features into the page templates, but not into the pages generated by partly running the edited application. Therefore, applicant submits that claim 1 is separately patentable over claims not

reciting editing features, such as claims 22, 51, 74, 90, 114 and 125, and also over claims not reciting operating via editing features, such as claims 6, 26and 59.

b) Arguments

As explained in sections A.ii above and C.i below, the WebWriter Editor does not run the edited application during editing. In stark contrast, applicant's editor does. Claim 1 is considered patentable over the cited combination at least because (1) the recited document generator runs at least part of the edited application by generating documents that include editing features, and (2) the recited editor operates via editing features incorporated into said generated documents. This is recited in Claim 1 as:

> a document generator program running at least part of one of the applications being edited and generating the generated documents, said generated documents including additional editing features for interpretation by the browser program; and
>
> an editor program dynamically operating on the generated documents displayed by the browser program via the editing features.

The final office action refers generally to the description of the WebWriter Page Generator in the WebWriter I article (see Office Action dated Jan 21, 2010 at p. 3) as prior art for the page generator, but provides no specific reasoning to support the rejection. There is no teaching or suggestion in either WebWriter article that the WebWriter Page Generator produces pages having editing features, even though the section labeled "The Web Writer Page Generator"on page 9 of the WebWriter I article gives a detailed description of the WebWriter Page Generator and does not mention editing features. Thus, the Examiner is mistaken regarding his conclusion that WebWriter I discloses a page generator as claimed. In fact, this would not make any sense because the WebWriter Page Generator does not run during editing – it runs only after editing, as explained in sections A.i, A.ii and C.i. Because the documents generated by the WebWriter Page Generator do not contain editing features but the claim requires "said generated document including additional editing features" the claim is patentable over the cited art.

c) Traversing The Examiner's Reasoning

The examiner also specifically referred to the phrase "on the fly" in his reasoning, which is explained on p. 9, col. 1 of the WebWriter I article. (See p. 3 of the final Office Action).

However, it appears the Examiner is trying to associate a meaning for this phrase that is not consistent with its use and description in the article. The article discusses the internal working of the page generator, i.e., the use of the on-the-fly parsing technique, **at runtime**, to replace dynamic areas while traversing the page template character by character. This is discussed in more detail in section C.i on page 32.

For the editor program, the examiner cites page 9, column 1, section entitled "Web Writer Page Generator." However, the cited portion describes the WebWriter Page Generator, and does **not** describe any editing functions. In an interview with applicant, the examiner emphasized the following text from the reference: "*WebWriter then adds a placeholder for the output area [which] will be replaced **at runtime** by the output of a script . . . .*" (*see* p. 4, 2nd ¶, under heading "Placing Dynamic Areas", emphasis added), and stated that he interprets the replacement of the placeholders at runtime as being an editing operation. Applicant totally disagrees. The WebWriter page generator transforms a page template into a generated page for display by a web browser, thereby replacing dynamic areas with the output of scripts. This process is usually called template expansion or running of a dynamic web application. It is never interpreted as an editing process, because it does not permanently modify the page template, but just temporarily transforms the page template into a page for display, and later discards the page. Also, in editing, there is usually an explicit user interaction that requests that a modification be made. The replacement of dynamic areas in the WebWriter page generator, however, is a totally automatic process taking place as an implicit step in executing an application (and not during editing). Finally, in the context of the claim, editing means editing of software applications, and that is different from transforming a page template into a final page, something that is done by a page generator. In addition, even if the replacement at runtime operation was interpreted as editing, which applicant disputes, this does not teach or suggest that the WebWriter page generator generates pages having editing features as claimed, nor does it teach that the WebWriter editor operates via the editing features.

It seems more natural to use the WebWriter Editor as prior art for applicant's editor, rather than the WebWriter Page Generator, as the examiner did.

The examiner appears to have cited WebWriter II only to incorporate the explicit teaching of a server computer and a web server. This, however, has no influence on the working of the WebWriter Page Generator and thus, Claim 1 is patentable over the cited combination.

The examiner also states "see page 2, first paragraph for buttons and computing content on the fly for components as claimed in claim 59." This comment is irrelevant for claim 1 (and for claims 26, 59 and 90) since these claims do not recite components and do not have any limitations on components.

d) Claim Limitations Not Addressed

The phrases "editor program operating on the generated documents" and "via the editing features" are key structural limitations of the claim language that were not specifically addressed by the Examiner, and which connect the running application to the editing process. The claim recites that the editing features are part of the generated documents, and the preamble states that the generated documents are displayed by the browser. The editor is then coupled via the editing features, and *voila*, you are editing a running application in the browser (please also refer to section C.ii).

e) Traversing The Examiner's Arguments from Office Action dated July 1, 2009

In the final Office Action, the Examiner did not repeat or incorporate by reference the arguments presented in the Office Action dated July 1, 2009. Instead, these arguments were replaced by a single new argument which is discussed in the Overview section A.i. Applicant addresses the prior arguments here just in case the examiner chooses to come back to his previous arguments. If not, then this section may be skipped.

On page 23 of the Office Action, the examiner states that the plain language of claims 1, 6 and 22 "*doesn't preclude, include or exclude when execution or running of the editor as taught by Web Writer is performed.*" Applicant respectfully disagrees. In fact, claim 1 specifically requires that the document generator runs at least part of one of the applications and generates generated documents, the preamble says "applications generates generated documents" and that the generated documents include additional editing features. This makes it clear that the generated documents that result from running the application include editing features. In this way, editing and running can take place simultaneously.

In contrast, WebWriter discloses two different programs that run at different times. The first program is the WebWriter editor, which includes its own page generation part that generates

pages with editing features, but does not run the edited application. The second program is the WebWriter page generator, which is not used during editing. but after editing, and which runs the application but does not insert any editing features. In contrast, claim 1 requires a single page generator that runs the application and, while doing so, editing features are inserted into the generated pages. Claim 22 in turn requires the editor program to operate on generated documents, whereby the generated documents result from a document generator executing the components. This also requires execution of components during editing because the editor operates on the output of the executing components.

Applicant notes that claim 125, for example, nicely recites running during editing by requiring a method for editing an application comprising the steps of running at least part of the application. Although the examiner's argument on page 23 of the Office Action is explicitly limited to claims 1, 6 and 22, his reasoning for claim 125 also refers back to the reasoning of claim 1.

The examiner further states on page 23 that "Running saved content such as a template on the fly, in one or more steps isn't disclosed as not meeting Applicant's claimed limitations as recited in Applicant's claims." This statement appears to be a misinterpretation of applicant's argument. Applicant refers to p. 6, col. 2 of WebWriter I, in the section entitled "Running the Application:" "***Once*** *a Web Writer application is created and* ***saved*** *to disk it can be* ***run*** *in one or two ways*" (emphasis added). In applicant's reading, this sentence makes it clear that the web writer application runs **after** it is saved to disk, and the use of the word **once** at the beginning makes clear that the application does **not and cannot run before** it is saved to disk. In contrast, applicant's editor runs the application during editing, i.e., also before the application is saved to disk. As explained above, claim 1 requires running of the edited application during editing by requiring generated pages to have editing features.

Further, on page 23 of the Office Action, the examiner discusses "*features upon which applicant relies (i.e. not running only after editing and only running during editing)*" (See last full paragraph on page 23). In fact, applicant relies on the feature of running the edited web application during editing. Applicant never stated that "only running during editing" is a distinctive feature. In fact, applications edited with applicant's editor keep running after editing as well. Applicant also never stated that "not running only after editing" is a distinctive feature. In fact, this phrase was introduced by the examiner and is very unclear. Applicant stated that

"WebWriter runs the edited application only after editing," which means that WebWriter runs the application after editing but does not run the application during editing. In contrast, applicant's editor runs edited applications during editing and several claims require this as explained above.

The examiner seems to have taken applicant's statement that "WebWriter runs the edited application only after editing" and negated it into "not running only after editing" and then assumed applicant was relying on this as a distinctive feature. In fact, the distinctive feature is best characterized as "running during editing." The recitation of this feature in the claims has been discussed above.

Since the examiner made incorrect assumptions about the distinctive features, his reasoning about these features in the claim language is flawed and inapplicable.

However, the examiner further states that WebWriter does in fact disclose this limitation. It appears that the examiner is reasoning that "this limitation" refers back to "only running during editing." Applicant strongly disagrees. Editors, including both the WebWriter editor and applicant's editor, produce applications that run after editing. An editor that runs the edited application only during editing would not be useful because it is the main task of an editor to produce applications that run at least after editing.

In the first paragraph of page 24 of the Office Action, the examiner argues "it is actually quite inherent that the button is an executable component and hence this is being performed during editing", without providing any evidentiary support. In fact, applicant believes that buttons of the edited application in WebWriter are deactivated during editing and become working only after saving and running the application, because otherwise clicking a button would interfere with the editor. Even more importantly, however, a button is just an HTML element and not a server side component or application. Claim 1, for example, explicitly says that it's a server side application running during editing, and Claim 6 recites server side components and "instructions contained in said selected component on the server."

The examiner then cites the 2nd paragraph of the WebWriter I abstract: "WebWriter includes a direct manipulation Web page editor, the WebWriter Editor, which runs in the web browser as a CGI-service, and the WebWriter Page generator, which creates new pages as the application runs". The examiner states that the WebWriter page generator runs the application. Apparently, the examiner assumes that the WebWriter editor and WebWriter page generator

work together during editing. This is, however, not the case. They are both separate programs, and the WebWriter page generator is used only after the edited document is saved, as discussed above. There is no support anywhere is either WebWriter article for the Examiner's assumption that the WebWriter editor and the WebWriter page generator work together during editing.

Finally, the examiner states "in page 4 of Web Writer, under the Heading, "Creating the Template pages", Web Writer is said to be an editor that runs in a Web browser, ***which seamlessly generates/creates web pages while the application is run***." (emphasis added). The underlined portion of this sentence **is not actually included** in the prior art reference, but was added by the examiner, and there is no support for such a statement in either WebWriter article.

ii. Components

Applicant's system allows the developer to create web applications by plugging together software components on document templates using the inventive editor. These special type of components contain instructions, are executable on **the server side by the document generator**, and dynamically **generate browser code**. In applicant's preferred embodiment, components are implemented as objects of an object oriented programming language.

With respect to claim 1 the examiner writes "buttons and computing content on the fly for components".

HTML buttons are implemented **client side** and therefore **executed by the browser** on the client. Also, buttons **do not generate browser code**. Therefore, buttons are not components in the sense of the claim language because the claims have additional limitations. For example, claim 2 says "the document generator executes selected components." In contrast, buttons are executed by the browser. Claim 6 says "component executing first instructions contained in said selected component on the server," but buttons are executed on the client. Claim 51 says "components containing instructions to generate browser code," but buttons do not generate browser code.

According to WebWriter I at p.9, section entitled "The WebWriter Page Generator," the content computed on-the-fly actually is computed by scripts associated with dynamic areas. The WebWriter I article defines dynamic areas as locations within a template page (WebWriter I at p.2 section entitled "The WebWriter Application Model") "*template page, which includes the specification of locations within the page, called dynamic areas . . . .*") and adds that a script is

specified for each dynamic area which is executed at run-time (p.2 "*dynamic areas, where computed content should be inserted at* ***run-time*** *. . . a script that* ***will*** *be run to provide to provide the computed content for that dynamic area*" (emphasis added)). Applicant therefore assumes that the examiner interprets scripts specified for the dynamic areas as prior art for the components recited in applicant's claims.

iii. Claim 6

a) Grouping

Independent Claim 6 should be considered separately and apart from the other claims. Claim 6 cannot be grouped with other claims because this claim requires the important distinctive feature, namely: "the component has a similar appearance as on the generated page …"

Even for someone having developed an editor which runs at least part of one of the applications being edited, as in claim 1, it would not have been obvious to invent an editor for editing selected components on page templates, wherein a component has an appearance similar to that on the generated page, as required by claim 6. For example, one might run components to provide interface information about the components to an editor. Vice versa, someone having developed an editor wherein a component has an appearance similar to that on the generated page might use other means to achieve the required appearance, e.g., by display routines built into the editor for some specific components.

In addition, applicant refers to the reasoning above for the separate patentability of claim 1 with respect to components in general. Separate patentability with respect to the remaining claims is discussed below.

b) Arguments

As explained in section A.ii, WebWriter replaces dynamic parts of pages with placeholders during editing. The placeholders show the script name and a formatting string, and so the displayed appearance during editing usually looks very different from the appearance of a component on a generated page. In contrast, claim 6 requires that the components have a similar appearance as on the generated page. Therefore, applicant submits that the claim is patentable over the cited combination.

The examiner did not include specific reasoning for the rejection of claim 6, but instead relied upon his reasoning as applied to claim 1. However, Applicant submits that this reasoning is simply not applicable to claim 6, because claim 6 has limitations on component appearance that are not recited in any other claim and not taught by the cited prior art.
Further, the failure to provide detailed reasoning amounts to a failure to make a *prima facie* case of obviousness.

c) Traversing The Examiner's Arguments from Office Action dated July 1, 2009

Although not repeated or incorporated by reference in the final Office Action, the examiner made arguments in the Office Action dated July 1, 2009 that are addressed here by applicant. On page 25, after the headline "Response(2)" the examiner argues that merely adding the limitation of having similar appearance is not distinctive. Applicant strongly disagrees, and submits that WYSIWYG editing is in fact a major improvement in the field of editing technology, as further discussed at the end of section A.ii.

On page 25, the examiner also refers to the terms "edited template in WebWriter which contains both the static and dynamic portions" and "the created pages which are run in browser" and refers the 2nd paragraph on page 2 of the WebWriter I reference. Unfortunately, the cited portion neither contains these terms nor seems to discuss anything related. Apparently, the examiner believes that both views (editing and running) appear similar in WebWriter I. This is simply not true, however, as argued above in section A.ii, or by reference to Figures 6(b) and 7(b) or Fig. 10(b) and Fig. 11 of WebWriter I, which show that the respective views do not appear similar.

iv. Claim 22

Independent Claim 22 should be considered separately and apart from the other claims. In particular, claim 22 does not belong into the same group as claim 1 because claim 1 does not recite components at all, while claim 22 requires components and executing of the components by the document generator that produces pages which the editor operates on.

Someone having developed an editor which runs at least part of one of the applications being edited, as in claim 1, would not necessarily use components to build applications. Vice

versa, executing components by the document generator does not necessarily put editing features into the generated documents as required by claim 1.

Applicant also refers to the reasoning for separate patentability of claim 1 with respect to components and editing features. Further, applicant refers to the reasoning for claim 6 and the discussion of the similar appearance of components. Separate patentability with respect to the remaining claims is discussed below.

As explained in sections A.i and A.ii, the WebWriter Editor does not execute part of the edited application during editing. As described in section C.i and in WebWriter I at page 9, the scripts contained in dynamic areas are executed after editing, not during editing, by the WebWriter Page Generator.

Claim 22 requires "*an editor program operating ... on generated documents*" and "*a document generator ... for generating **the** generated documents*" (emphasis added). The use of the article "the" before the term "generated documents" makes it clear that the editor is operating on the documents generated by the document generator. This is not what is taking place in WebWriter. The WebWriter Page Generator generates documents **after editing**, and so there is no chance for the WebWriter Editor to operate on the generated documents since the generated documents do not yet exist at the time of editing. Instead, the WebWriter Editor has its own page generator which is different from the WebWriter Page Generator. For that reason, claim 22 is patentable over the cited combination.

Alternatively, the examiner could have cited the page generator inside the WebWriter Editor as prior art for a document generator program in claim 22. However, this fails as well, because claim 22 requires "*a document generator for executing said components*" and as explained in sections A.i, A.ii and C.i, the WebWriter document generator executes the scripts of dynamic areas, but the WebWriter editor does not.

In summary, claim 22 requires an editor operating on documents generated by the document generator that has instructions for executing components, and so requires that the components are executed during editing. This is simply not the case in WebWriter, and therefore applicant submits that the claim is patentable over the cited combination.

The examiner did not include specific reasoning to support his rejection of claim 22, but relied on the reasoning applied to claim 1. However, applicant submits that this reasoning is not applicable to claim 22 for the reasons just given with regard to separate grouping and

patentability. Since the Examiner has failed to provide detailed reasoning, as is required, he has not made a *prima facie* case of obviousness.

v. Claim 26 and Claim 32

Independent Claim 26 and dependent claim 32 form a group of claims that should be considered separately and apart from the other claims. Claim 26 cannot be grouped with any claim not in claim set 26 or 59 because claim 26 requires the important distinctive feature that "*the second document appears and functions similar to the run-time view of the first document*" and because this feature is not recited in any other claim except claim 59.

Note that claim 6 and claim 26 cannot be grouped for the same reason. Although claim 6 requires similar appearance of a component, claim 26 requires both similar appearance and function for the document. Similar function is a different distinctive feature that is not-obvious in the light of similar appearance. Further, an editor that is constructed according to claim 1 does not necessarily generate a second document that appears and functions similar to the run-time view of the first document and such a feature is not obvious.

In other words, with respect to claim 1 or claim 22, applicant submits that an editor that is constructed according to claim 1 or 22 does not necessarily generate a second document that appears and functions similar to the run-time view of the first document. It is possible to run part of the application as in claim 1, or to execute components as in claim 22 during editing, e.g. to provide interface information on components to the editor, but not necessarily to make the generated documents appear similar or function in a similar way. It is also possible and useful that the generated documents have a similar appearance but not a similar function. Although applicant prefers that the generated documents do function during editing, because this allows testing of the application during editing, "similar function" is actually a distinct new idea, and someone having invented an editor providing similar appearance during editing would not necessarily invent an editor that provides "similar function" as well. Also, similar function requires various provisions, e.g., in the client part of the editor, that are not required to provide a "similar appearance". With respect to claim 6, the same arguments of "similar appearance" versus "similar function" apply since claim 6 just requires similar appearance. In addition, applicant refers to the reasoning for separate patentability of claims 1, 6 and 22, since claim 26 like claim 1 does not require components, but claim 6 and claim 22 do.

Claim 26 describes a system to modify a first document stored on a server whereby a second document is displayed to the user that appears and **functions** similar to the first document. Applicant emphasizes the fact the recited elements of the claim are located on the server, wherein the server comprises:

> a document store;
>
> a first software program including instructions for transforming at least one first document into a second document having features which permit editing of the first document such that at least a part of the second document appears and functions the same as the first document; and
>
> "*a second software program including . . .* instructions *to modify the first document,*

Claim 26 stands rejected on the basis of a combination of WebWriter I and WebWriter II. As explained in detail in sections A.i and A.ii, WebWriter does not execute the application being edited during editing. Therefore, it is clear that the application being edited in WebWriter does not function at all during editing. In contrast, claim 26 requires "*at least a part of the second document appears and* ***functions*** *similar to the first document.*"

In fact, WebWriter transforms a first document being edited into a second document which permits editing of the first, thereby trying maintain the appearance of the static parts of the document. However, as discussed in section A.ii, WebWriter does not make any effort to maintain the appearance of the dynamically generated parts of the document or of the functionality of the document.

The examiner did not include specific reasoning to support his rejection of claim 26, but again relied on his reasoning as applied to claim 1. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

For all these reasons Claim 26 is patentable over the cited combination.

Claim 32 depends from Claim 26 and for all the same reasons is patentable over the cited combination.

vi. Components may Cooperate with the Editor

Just as for WebWriter's scripts of dynamic areas, Applicant's components are able to run at run-time. However, applicant's components are run and used during editing as well, and this feature is not taught or suggested in either WebWriter article or by a combination of the articles. In fact, applicant's components cooperate with the editor during editing in various ways, e.g., by

generating browser code for display of the component inside the editor, by drawing handles, or by collecting and providing information about the component to the editor. While the components' normal instructions for generating browser code are used both during and after editing, the components also have features to cooperate in editing, e.g., instructions to position the handles.

In contrast, the scripts specified for WebWriter's dynamic areas are not used at all during editing, but the editor replaces the dynamic areas with placeholders. ("*WebWriter then adds a placeholder for the output area [which] will be replaced* ***at runtime*** *by the output of a script . . .*" *see* page 4, 2nd para. under heading "Placing Dynamic Areas", emphasis added)

In fact, applicant submits that a further distinctive feature is recited by the claim language "components cooperating with the editor during editing." The present Office Action does not cite any specific portion of WebWriter as disclosing this feature, because of course it is not disclosed. WebWriter's dynamic areas call independent Unix scripts at runtime, as explained on p. 9 at the end of column 1 and top of column 2: "*When WebWriter encounters an OUTPUT tag it ... runs the specified program on its arguments ...*". The WebWriter Editor cannot be extended by any external components. This is explained in WebWriter I on page 5 column 1 Section entitled "Script:" "*The user selects from a small set of* ***built-in*** *modules ...*" (emphasis added). It is only the WebWriter Page generator that can call arbitrary scripts and more specifically scripts of dynamic areas at run time. Also there is nothing in the scripts that in any way facilitates editing of the scripts.

This distinction is recited in the claim language of claim 51 ("*each component having features to cooperate in editing the component*"), claim 74 ("*the components having the ability to cooperate with the editor*") and of claim 114 ("*the components including first features adapted to cooperate with an editor in dynamically editing said component.*")

vii. Claim 51 and Claim 52

Claim 51 and dependent claim 52 form a group of claims to be considered together but not with any other claims. Claim 51 is an independent claim describing a system for editing components on web document templates. Claim 51 cannot be grouped with any claim not in claim set 51, 74 or 114 because claim 51 requires the important distinctive feature of "each component having features to cooperate in editing the component." Since the scripts of dynamic

areas of the prior art are used at runtime only and do not have any means to assist in the editing process cooperation of components in editing is a non-obvious improvement of the editing process. For all these reasons, and for the individual reasoning discussed with regard to claims 1, 6, 22 and 26, claim 51 stands on its own and does not fall together with any other claim not in claim set 51, 74 or 114.

Someone having developed a system according to claim 22 would not necessarily use specific components with features that cooperate in editing, as required in claim 51, but instead might have wanted to use existing components without such features. Vice versa, someone having developed a system with components having such features would not necessarily also execute the components by the document generator program as required in claim 22.

The examiner did not include specific reasoning for claim 51, but once again relied on his reasoning for claim 1. Applicant submits that this reasoning is not applicable because claim 51 includes the recitation of ***components having features to cooperate in editing the component*** while claim 1 does not. As explained in section B.vi, there is no cooperation between the scripts associated with the dynamic areas or the content computed on-the-fly and the WebWriter Editor.

Therefore, applicant submits that the claim is patentable over the cited combination. Also, because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

Claim 52 depends from Claim 51 and for all the same reasons is patentable over the cited combination.

viii. Claim 59 and claim 71

Claim 59 is an independent claim directed to a software development system for dynamic web documents. Claim 59 and dependent claim 71 form a group that should be considered separately and apart from any other claim. Claim 59 cannot be grouped with any claim not in claim set 26 or 59 because claim 59 has the distinctive limitation "dynamic web documents which look and function similar to the end user's view of the documents" which is not required by any other claim except Claim 26.

However, Claim 59 should be considered separately and apart from any other claim not in claim set 59 including claim 26 because claim 59 also requires the editor to have instructions

for requesting that the document generator process a dynamic document. These instructions are a feature not recited in any other independent claim.

Someone having developed a system according to claim 26 would not necessarily have an editor as explicitly recited in claim 59, where the editor has instructions for requesting that the document generator process a dynamic document. For example, applicant's editor has a client part running in the browser using the javascript functions to request the page generator to process a dynamic document, while WebWriter I does not have a similar client part, but instead just generates pages with an appropriate HTML form.

Also, claim 59 does not recite any components and therefore does not fall together with any claim that requires components for the same reasons as in claim 1, and as reasoned with regard to the individual claims such as 6, 22, 51, 74, 114 and 125.

As explained above in sections A.i, A.ii and C.i, the inventive editor is capable of running a dynamic web document being developed during editing, and so the dynamic web document appears functional during editing. This distinction is expressed by the following claim language: "*generated documents ... which look and function similar to the end user's view of the documents"* and by *"the editor program comprising first instructions for requesting the document generator to process a dynamic web document leading to a generated document.*" This language makes clear that the editor indeed runs the dynamic web document during editing, by requesting the document generator to process a dynamic web document. Additional claim language, namely "*instructions to modify* ***the dynamic web document***" make clear that the dynamic web document that is being edited is the one being requested.

The examiner did not include specific reasoning for claim 59 but yet again relied on his reasoning as applied to claim 1. (see final office action at p. 3). This reasoning, however, includes a comment that is at best not very clear, namely: "page 9, column 1, see WebWriter page Generator, see on the fly, also see page 2, first paragraph for buttons and computing content on the fly for components as claimed in claim 59." Since claim 59, like claim 1, does not recite any components on document templates, applicant does not believe that the examiner's comment regarding components is applicable for claim 59. The mentioned WebWriter Page Generator runs after editing as explained in sections A.i, A.ii and C.i, and so it is clear that the WebWriter Page Generator does not receive requests from the WebWriter Editor during editing. In contrast, claim 59 requires *"the editor program comprising first instructions for requesting the document*

generator to process a dynamic web document leading to a generated document." For all these reasons, applicant submits that claim 59 is patentable over the cited references.

Claim 71 depends from Claim 59 and for all the same reasons is patentable over the cited combination.

ix. Nested Components and a Varying Set of Components

"Nested Components" is a distinctive feature used in some of the dependent claims, while the "varying set of components" is recited in the independent claim 74 and dependent claim 5.

WebWriter's dynamic areas are represented using a special <output> tag as explained on page 9 column 1. The output tag contains a formatting string. As explained at the end of column 1 and top of column 2 of page 9 "*When WebWriter encounters an OUTPUT tag it extracts the SCRIPT field and formatting string, runs the specified program on its arguments, formats the results using the formatting string and writes the result to the output.*". There is no teaching or motivation that the formatting string could contain itself nested OUTPUT tags.

On page 28 of the Office Action dated July 1, 2009, the examiner argues that a page template containing a component is a nested component. However, page templates are not components in the sense of claim 4 (or 81 and 87) because base claim 2 (or 74) requires page templates containing components. So, only items that can be placed on page templates are seen as components in the sense of claim 2, and page templates cannot contain other page templates, so page templates are not components in the sense of the claim.

In contrast, applicant's components can be nested. Inside a component on a document template, HTML code and other components can be denoted. An enclosing component can generate browser code for insertion at the begin tag and the end tag, and it can also dynamically control insertion of content into the final document. For example, the component can hide its content, insert it once, or multiple times. Thus, the components nested inside may dynamically be hidden, or multiple instances of a component may be displayed. In this way, the number and kind of components on a generated page is no longer static but can dynamically change at run time.

This is denoted using the claim language of independent claim 74: "*the set of components on the generated documents can vary for different document requests for said document template;*" and, for example, dependent claim 4: "*wherein at least one of the components*

contains at least one other component;" and finally claim 8: "*a component is nested within a component*".

Note that not only do applicant's components have these features, but also applicant's editor has the ability to handle components with these features and, for example, has operations to place one component inside another one, or to exclude or display multiple instances of a component during editing, simply by executing the application during editing.

x. Claim 74

Claim 74 is an independent claim which stands on its own and does not fall together with any other claim. Claim 74 requires the feature that "the set of components on the generated document can vary for different document requests . . . ." This is a distinctive feature not required by any of the other claims, except dependent claim 5, and therefore claim 74 stands on its own and does not fall together with any other claim not dependent on claim 74 or claim 5. It is possible to develop many applications having a static set of components, but using a dynamic set of components gives more flexibility to develop more applications. Therefore, the dynamic set of components is a non-obvious additional feature which makes claim 74 separately patentable over all other claims except claim 5 and claims dependent on 74. Claim 5 is discussed further below.

Claim 74 is an independent claim describing a software development system for document templates and stands rejected as obvious based on the combination of WebWriter I and WebWriter II. In the Final Rejection, the Examiner did not include specific reasoning for claim 74 but handled claim 74 together with claim 1. Claim 1 does not require "components". Therefore, applicant submits that the examiners reasoning given is not applicable, does not discuss the distinctive features, and is therefore irrelevant for claim 74. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

Applicant believes that claim 74 is patentable over the cited art because it requires components cooperating with the editor. As explained in section B.ii, WebWriter I and WebWriter II do not teach or suggest any sort of cooperation between components and the editor. In contrast, claim 74 explicitly requires "*the components having the ability to cooperate with the editor*".

Applicant also believes that claim 74 is patentable over the cited art because it requires "*wherein the set of components on the generated documents can vary for different document requests*". In WebWriter, each dynamic area is filled by its script exactly once per request, as explained in section "The Web Writer Page Generator" on page 9 of the WebWriter I article. It is not possible that a script associated with a dynamic area is excluded from generation, or is generated multiple times, and so the number of scripts of dynamic areas executed per template page is fixed. In contrast, the claim requires that the set of components on the generated documents can **vary** for different document requests. Also see section B.ix for information on the varying component set.

For all these reasons, applicant submits that claim 74 is patentable over the cited references.

a) Traversing The Examiner's Arguments from Office Action dated July 1, 2009

The examiner did not incorporate these arguments in the final office action, so possibly they are outdated and need not be addressed by applicant. However, if the examiner comes back to these arguments, this section becomes relevant:

The Office Action dated July 1, 2009 contains an argument about claim 74 on page 26. The examiner maintains that the following is taught by the cited art: "*the set of components on the generated documents can vary for different document requests for said document template and components having the ability to cooperate with the editor.*" The examiner states:

> "In Web Writer on page 2 of his disclosure, the editor cooperates with the browser as well as the page generator seamlessly allowing the user to interactively construct application without the need to learn HTML as disclosed in page 2, first paragraph of Web Writer I."

This statement appears totally unrelated to the feature of components cooperating with the editor, or a varying set of components. In claim 74, components are defined as items being denoted on page templates. Neither the browser nor the page generator are such a kind of component.

In addition, applicant disagrees with the examiners statement listed above. Although the examiner's statement appears to be unrelated to claim 74, it may be of relevance to other claims.

Therefore, applicant notes that, contrary to the Examiner's assertion, the cited portion does not disclose or suggest any cooperation between the WebWriter editor and the WebWriter page generator. In fact, there is no mention whatsoever of the page generator in the cited portion of the reference, but it does say: "To create template pages, the designer uses the WebWriter Editor . . . ." It also says the final steps of application building are "saving and loading a stack and running the application," but there is no indication at that point what causes the application to run. Moving forward to page 6 of the reference, it says under the headline "Running the application" that "Once a WebWriter application is created and saved to disk, it can be run in one of two ways," and here is where it becomes clear that the WebWriter Page Generator runs the application, not the WebWriter Editor. Thus, the editor and the page generator have separate and distinct, clearly defined roles, and they do not cooperate to edit and run a web application.

xi. Claim 90 and claim 92

Claim 90 is an independent claim directed to an editor for use with a web browser. Claim 90 and dependent claim 92 form a group that should be considered separately and apart from any other claim. Claim 90 cannot be grouped with claims 6, 22, 51, 74, 114 and 125 because those claims require various features based on components, as discussed with these claims individually, while claim 90 does not. Also, the see reasoning about grouping and components for claim 1.

Claim 90 should be considered separately from claims 1, 26 and 59 and all other claims not dependent on claim 90 because claim 90 also requires that "scripts contained in said documents remain running during editing." The features of "running at least part of one of the applications" by the document generator (claim 1) and "the editor . . . requesting that the document generator program processes a dynamic web document" (claim 59) involve execution in the document generator and do not teach or suggest that browser side scripts remain running as well. Claim 90 however requires "scripts contained in said document remain running during editing." Also, the phrase "function similar" in claims 26 and 59 does not necessarily refer to scripts only.

In addition, claim 90 requires an additional first software program for execution within the browser, which is not recited, suggested or taught by any of the other independent claims. For all these reasons, applicant submits that claim 90 is separately patentable over all the other

independent claims. Finally, unlike most of the other independent claims, claim 90 does not recite document generation or a document generator.

Independent Claim 90 is directed to an editor embodiment. The editor is used with a web browser and allows a user to edit a document being actively displayed by the browser "wherein scripts contained in said document remain functional during editing." Further, the editor includes a client part, e.g., a first software program for execution within the browser that initiates editing functions when the user clicks on the displayed document.

The Examiner did not include specific reasoning for claim 90 but included claim 90 with his reasoning for claim 1. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness. In the arguments section of the Office Action dated July 1, 2009 on page 27, the examiner cited "specifications of locations within the page, called *dynamic areas,* where computed content should be inserted **at runtime** to produce the final page that will be displayed to the user" (emphasis added). In applicant's reading, the cited portion makes it very clear that the insertion is taking place at **runtime** and not at editing time. In addition, the portion refers to **computed** content which is inserted at runtime. It does not disclose **edited** content as argued by the examiner, and it is not prior-art for "edit" as required by claim 90. In this context, the term "inserted" does not refer to an editing operation, but instead to an internal operation of the page generator taking place at runtime while executing the application. The result of the insertion is a temporary page being displayed in the browser, while an editing operation typically produces a document for storage as a template page, not for temporary display. The cited portion therefore does not disclose editing at all, as required by claim 90.

As explained in sections A.i, A.ii and C.i above, WebWriter does not run the application during editing. In contrast, claim 90 requires scripts contained in the document to remain functional during editing.

In addition, claim 90 recites a first software program for execution within the browser. WebWriter I does not have such a program, since it does not seem to generate pages with client side scripts. WebWriter II does have client side scripts, however, as explained in section C.iii above, the combination of WebWriter I and WebWriter II is improper since they employ different architectures.

For all these reasons, applicant submits that claim 90 is patentable over the cited combination.

Claim 92 depends from Claim 90 and for all the same reasons is patentable over the cited combination.

xii. Claim 114 and Claim 128

Independent Claim 114 should be grouped together with dependent claim 128 and not with any other claims. Claim 114 cannot be grouped with any claim not in claim set 51, 74, or 114 because this claim requires the important distinctive feature of components having features to cooperate with the editor, and claims 51, 74 and 114 are the only claims reciting this feature. However, claim 114 cannot be grouped with claim 74 because of the dynamic component set limitation of claim 74. Further, although Claim 114 shares the cooperation feature with claim 51, both claims should not be considered together because claim 51 also requires "*a second software program transmitting, while processing selected requests, second documents to the first software program that make the first software program display a user interface for editing functions used for maintaining components on document templates*". The second software program is a non-obvious additional feature.

Independent Claim 114 was rejected without any specific discussion in the grouping of claims described on page 3 of the final Office Action. For this reason, applicant submits that the Examiner has failed to make a *prima facie* case for obviousness.

Claim 114 is an independent claim describing a system for editing components on web document templates. Claim 114 requires "*the components including first features adapted to cooperate with an editor in dynamically editing said component.*" Claim 1 does not include such a feature, and it does not even require "components". Therefore, applicant submits that the examiner's reasoning is not applicable and therefore irrelevant for claim 114.

Claim 114 requires "*the components including first features adapted to* ***cooperate*** *with an editor"* (emphasis added). As reasoned in detail in section B.vi above, WebWriter does not teach or suggest components that cooperate with the editor. Applicant submits that therefore claim 114 is patentable over the cited combination.

Also, despite the Examiner's comments to the contrary in the Advisory Action dated November 30, 2005, a part of the editor cannot be interpreted as a component because the part

lacks first features to cooperate with the editor in editing this part itself as required by the claim language "*components including first features to cooperate with an editor in editing said component.*" In addition, the use of the word cooperate makes clear that the components are not part of the editor.

Claim 128 depends from Claim 114 and for all the same reasons is patentable over the cited combination.

xiii. Claim 125 and Claim 127

Claim 125 is the only independent method claim describing a method useful for editing an application. Claims 125 and dependent Claim 127 form a group that stands on its own and does not fall together into a group with any other claim. Claim 125 requires selecting a component by clicking on selected portions of said view and the distinctive feature of identifying the selected component in the source code of the application. Both features are not recited in any other independent claim (except for the selecting by clicking in dependent claim 58), and for that reason alone, Claim 125 cannot be grouped with any other claim. In addition, claim 125 very clearly specifies running during editing by requiring the distinctive feature of "running at least part of the application" in a method of editing that application, in contrast to claims 6, 22, 26, 51, 74, 90, and 114. In addition, Claim 125 requires a method for editing with a step of executing components and can therefore not be grouped with claim 1, 26, 59 and 90 as these claims do not mention components at all. Claim 125 can also not be grouped with claim 6, 51, 74 and 114 because of the specific distinctive features discussed with these claims individually. For all these reasons, Claim 125 should be considered separately, and not with other claims or groups.

With regard to claim 125, the examiner argues a similarity to claim 1 in the Office Action dated July 1, 2009. However, Claim 125 includes method steps that do not relate to any limitation recited in claim 1. For example, Claim 125 requires "executing selected components" while claim 1 does not mention components at all; Claim 125 requires "selecting a component by clicking" while claim 1 does not mention clicking; claim 125 requires "identifying the selected component in the source code of the application" while claim 1 does not recite identifying, components or source code and most notably claim 125 recites running during editing by reciting a method for editing a document having a step of running at least part of the application.

Claim 125 stands rejected as obvious based on the combination of WebWriter I and WebWriter II. In the Rejection, the Examiner did not include specific reasoning for claim 125 but handled the claim together with claim 1. For this reason, applicant submits that the Examiner failed to make a prima facie case for obviousness.

Claim 125 requires a method useful for editing an application that is built using components and that operates by generating documents. The first step is "*running the application, thereby executing selected components.*" As has been extensively explained in sections A.i, A.ii and C.i and with regard to claim 1, neither of the cited WebWriter references perform this step during editing – they do not run the application and do not execute selected components during editing. Applicant submits that for these reasons, claim 125 is patentable over the cited combination.

Claim 127 depends from Claim 125 and for all the same reasons is patentable over the cited combination.

C. DETAILED DISCUSSION OF THE CITED ART WEBWRITER I AND WEBWRITER II

i. The WebWriter Page Generator and the Page Generation inside the WebWriter Editor are separate

WebWriter is a system for web application development having two distinct environments: a development time environment (the "WebWriter Editor") and a run-time environment (the "WebWriter Page Generator"). For example, the abstract of WebWriter I says:

> "This paper describes WebWriter an integrated system for constructing Web Applications . . . Webwriter includes a direct manipulation Web page editor, the *WebWriter Editor* . . . and the *WebWriter Page Generator* , which generates new pages as the application runs."

Both programs are described in a section called "Implementation," which is split into two subsections: "The WebWriter Editor" and "The WebWriter Page Generator." (see pp. 8-9). The WebWriter editor and WebWriter page generator are clearly separate programs, the editor being used during editing and the page generator being used at run time. The first paragraph of the section entitled "WebWriter Page Generator" on page 9 explains that the Web Writer Page

Generator is used at run-time, i.e. while an application created with WebWriter is running: "**When** a WebWriter application is **running**, each new page is assembled by the **WebWriter Page Generator**, **another** server based CGI C++ **program**..." (emphasis added) The use of the word "**another**" makes clear that these are indeed separate programs.

There is no explicit teaching in the article, much less a hint or suggestion, that the WebWriter Editor is using the WebWriter Page Generator. On the contrary, the sub-section called "The WebWriter Editor" describes the tree-based page generation method used by the editor (see pp. 8-9). The 2nd paragraph of the section says "*Each page **generated by the editor** contains HTML code that represents the appearance of the current document in the left column.*" During editing, the document is stored in the form of a content tree, and "*after executing each user specified action, WebWriter generates the next page by traversing the content **tree . . . .***" (page 8, last paragraph, emphasis added). This explains precisely and explicitly how page templates are displayed during editing, i.e., by traversing the content tree, and not by using the WebWriter Page Generator.

The examiner has specifically mentioned the phrase "on the fly" which is used in the implementation section of the WebWriter I page generator, but this phrase certainly does not mean what the examiner suggests. On page 9 column 1, the WebWriter I article states: "*The Page Generator generates a new page on the fly as it scans through the template page,*" and this just means that the WebWriter Page Generator is implemented using the "on the fly" technique, which is a common technique to implement parsers: It processes the document character by character and while doing so processes dynamic areas as soon as they occur in the input i.e. on-the-fly "*As it reads it copies each character to its output stream until it encounters ...* " (page 9 column 1, last full paragraph). The sentence explains the inner workings of the WebWriter Page Generator, but does not express in any way that the page generator is running during editing.

For all these reasons it becomes clear that the WebWriter Page Generator runs after editing and not during editing. Since the WebWriter Page Generator executes the edited application and the scripts of dynamic areas (as detailed in "The WebWriter Page Generator" section on page 9) it becomes clear that this also happens after editing and not during editing. And vice versa it is clear that editing features are produced only by the document generation inside the WebWriter Editor (as detailed in "The WebWriter Editor"on page 8) and not by the WebWriter Page Generator.

With respect to WebWriter II, the examiner only refers to this article for establishing prior art for "server" or "web server" on page 3 of the final office action. The examiner does not suggest that WebWriter II discloses running dynamic web run applications during editing.

ii. Editor Architecture

The classical editors described WebWriter I and WebWriter II have load and save operations. When loading a document, it is transformed into a content tree. Editing operations modify the tree. Finally, on a save, the tree is unparsed and stored into a file. During editing after each operation, the tree is traversed and displayed to the user. (See page 8 column 2 Headline The WebWriter Editor). In addition, there is the WebWriter Page Generator that, after the edited page has been saved, reads a file and executes scripts. This classical architecture requires that applications be executed only when editing is finished and the tree is sent to the server and saved to a text file, because the text file is needed for execution. During editing, documents are displayed inside the editor, i.e., inside an environment that considerably differs from the runtime environment, e.g., URLs, URL parameters, and the page generator are different. In order to successfully run an application, however, the document must run in environment that closely resembles the runtime environment.

In contrast, applicant's editor runs the application being edited in a first browser window and the client part of the editor in another window. The user interacts with the application running in the first window as usual. It runs in its normal runtime environment, e.g., using its normal URLs, URL parameters etc. and also uses its normal page generator. Therefore, it appears fully functional. While editing, additional editing features are included in the generated pages, in a way that does not interfere with the normal operation of the application. The editing features include handles. The user can click on a handle or interact with the editor window to issue a modification. Then, the client and server part of the editor perform the modification directly in the document template file. Afterwards, the editor asks the browser to reload the page in the first window, which causes the now modified application to be run by the page generator as usual.

iii. The Combination of WebWriter I and WebWriter II Is Improper

In rejecting applicant's claims, the examiner has relied on two prior art references describing two variants of the WebWriter Editor referred to herein as WebWriter I and WebWriter II. With respect to the independent claims, the examiner cited WebWriter II only to incorporate the explicit teaching of a server, while he recites much more significant portions of WebWriter II for some of the dependent claims.

WebWriter II is a newer development from the same authors, so one might assume that WebWriter II already contains as far as possible all the features of WebWriter I. This is not true. WebWriter II uses a different architecture called "meteor shower" which performs most of the editing operations on the client computer (except for loading the document before editing and saving the document after editing), while WebWriter I did most of the editing operations on the server. (*See* WebWriter II, sec. 1.2 entitled "Increasing interactive performance"). This new architecture makes WebWriter II work faster but teaches away from applicant's idea because running the edited server side application necessarily takes place on the **server**, while WebWriter II stores page templates as a tree on the **client** during editing. Applicant's editor does the editing in a distributed way, since many operations are done client side (e.g. displaying information about components and running client side parts of an edited document during editing) while others are done server side (e.g. modifying the edited document and running the server side parts of it during editing).

Because of the different architectures used in WebWriter I and WebWriter II, applicant submits that it is not obvious to simply combine WebWriter I and WebWriter II as suggested by the Examiner. Including a feature from one version into the other requires adaptation to another architecture (e.g., moved from server to client or vice versa), which may be complicated or even impossible. To pick and choose features from different prior art disclosures in order to form an obviousness rejection amounts to hindsight reasoning, which is impermissible. Applicant therefore submits that the combination is improper, and on that basis, applicant submits that the Examiner has failed to establish a *prima facie* case of obviousness.

D. THE DEPENDENT CLAIMS ARE PATENTABLE OVER THE COMBINATION OF WEBWRITER I AND WEBWRITER II

INTRODUCTION

Many of the dependent claims recite distinctive features thereby creating combinations that are considered separately patentable from the respective base claims, and therefore applicant submits that such claims do not fall together with the base claim. Many of these dependent claims should therefore be considered separately, as discussed below. On the whole, the dependent claims do not form a repetitive set of minor features, but in most instances, useful features are added to form unique combinations that are not merely repeated in other claims.

i. Claims 2-5 and 41-42

Claims 2-5 and 41-42 are dependent from Claim 1 and should be considered patentable over the cited combination for all the same reasons. However, these claims do not fall together with Claim 1 as discussed below.

a. Claim 2

Claim 2 is dependent on claim 1, but should be considered separately and apart from claim 1 and all other claims. Claim 2 further requires that "*the editor provides features to insert, modify and delete components on at least one document template*" and that the document generator "*executes selected components*" while the generated documents have editing features. Because of these distinctive features, claim 2 does not fall together with claim 1. Further, Claim 2 does not fall together with claim 22 and its dependent claims because claim 2 is dependent on claim 1and so requires that the editor operate via editing features while claim 22 does not recite editing features at all. Claim 2 also does not fall together with any claim from claim sets 6, 26, 51, 59, 74, 90, 114 and 125 because of the features discussed with regard to these independent claims above.

With respect to base claim 1, the Examiner inexplicably referred to the WebWriter page generator when reciting prior art for the claimed editor. (See final Office Action at p. 3) The WebWriter page generator, however, clearly does not have features to insert, modify and delete a component, as required by claim 2. Section C.i makes clear that it is the WebWriter Editor which inserts and modifies components, not the WebWriter page generator. For that reason alone, the

examiner's reasoning in is error and inconsistent, and thus the examiner again did not establish a *prima facie* case of obviousness.

Claim 2 together with its base claim 1 requires a single document generator, which generates documents having editing features and executes components. The page generator described in the WebWriter articles, however, is only running at run-time, after editing, and therefore executing scripts only at run-time. The editor described in the WebWriter articles generates editing features during editing only, and not at run-time, but does not execute components during editing as required by the claim. As explained in section C.i, the WebWriter Editor and the WebWriter Page Generator are two distinct programs running at different times. This is clearly distinct from a single page generator executing components and the generated document including editing features at the same time. Therefore, applicant submits claim 2 is patentable over the cited combination.

b. Claim 3

Claim 3 is dependent on Claim 2, and for all the same reasons, applicant submits that claim 3is not taught or suggested by the cited prior art.

1. Components React Interactively on User Input

Claim 3 includes the distinctive feature "*wherein at least one of the components reacts interactively on user input by executing instructions of said component on the server computer.*" Further, this distinctive feature is used only in the dependent claims.

Some of Applicant's components can not only display themselves during a first browser request, but also on a subsequent request, after user interaction, the components can process the data sent back from the browser to the server. These components are called interactive. Several of the dependent claims are based on the use of interactive components.

In contrast, the scripts called by WebWriter's dynamic areas are just used for processing of a single request. As clearly explained on p. 9 of WebWriter I, left column, heading The Web Writer Page Generator, the scripts are called during document generation to produce output for an output area. This takes place during processing of a single request. The scripts do not have a concept of multiple requests or subsequent requests and can therefore also not react in any way on a subsequent request.

This feature is also recited in claim 24: "*at least one of the components ... can react on subsequent document requests containing user responses by executing selected instructions of said component*."

2. Grouping

Claim 3 is dependent on claim 2, but should be considered separately and apart from the other claims. Claim 3 requires the important distinctive feature of components that react interactively on user input as described just above. No other claims recite this feature, except for claims 24 and 78, and therefore claim 3 cannot be grouped with any other claims except 24 and 78 and dependent claims 4 and 5. Grouping with these claims is discussed later.

3. Argument

With regard to claim 3, the examiner cited Fig. 2 of WebWriter I. However, the cited figure just shows the user interface of the editor. It does not show any dynamic areas and consequently does not give any information on components. In contrast, claim 3 requires "*at least one of the components that reacts interactively on user input by executing instructions of said component on the server*". As reasoned in section B.ix, above, WebWriter does not teach or suggest anything like interactive components. For all these reasons, applicant submits that claim 3 is patentable over the cited combination.

c. Claim 4

Claim is dependent on Claim 3, and for all the same reasons, applicant submits that claim 4 is not taught or suggested by the cited combination. However, Claim 4 does not fall with claim 3, and should be considered separately and apart from the other claims. This claim requires the important distinctive feature of nested components ("*wherein at least one of the components contains another component*") as described in section B.ix. No other claims recite this feature, except for claims 8, 81 and 87, and therefore claim 4 cannot be grouped with any other claim except claims 8, 81 and 87, and grouping of these claims is discussed later with regard to claims 8, 81 and 87.

With regard to claim 4, the examiner cites WebWriter I, Figs. 2 and 4. Fig. 4 shows a menu of available HTML tags, and does not give any information about dynamic areas. Fig. 2

also does not provide any information about dynamic areas. Claim 4, however, requires "*at least one of the components contains at least one other component*." Since components are apparently interpreted by the Patent Office as scripts associated with dynamic areas, both figures are then irrelevant for the showing one component containing another one. In the office action dated July 1, 2009, on page 28, the Examiner takes the fact that a page template contains a dynamic area as an example of one component containing another component. However, the base claim 2 already makes it clear that document templates contain components ("*document template capable of containing components*") and so only items that can be placed on page templates are components in the sense of base claim 2; and this is not the case for document templates. So, contrary to the Examiner's assertion, a page template containing a component is not an example of a nested component.

Section B.ix above explains that WebWriter does not teach or suggest one component containing another one. Therefore, claim 4 is patentable over the cited combination.

d. Claim 5

Claim 5 is dependent on claim 3, and for all the same reasons, applicant submits that claim 5 is not taught or suggested by the cited combination. However, Claim 5 does not fall with claim 3, and should be considered separately and apart from the other claims. Claim 5 requires the important distinctive feature of a varying set of components ("*wherein the set of components . . . can vary for different requests*") as described in section B.ix. No other claims recite this feature, except for claim 74. However, claim 74 also requires components cooperating with the editor, and for that reason, claim 5 cannot be grouped with claim 74.

With regard to claim 5, the examiner cites page 9 of WebWriter I. However, review of page 9 reveals that each dynamic area is filled by its script exactly once per request. It is not possible that a script associated with a dynamic area is excluded from generation, or generated multiple times, and so the number of scripts of dynamic areas executed per template page is fixed. In contrast, the claim requires that the set of components on the generated documents can vary for different document requests. Also, see section B.ix above for further arguments regarding a dynamic set of components. For all these reasons, applicant submits that claim 5 is patentable over the cited references.

e. Claims 41-42

Claim 41 is dependent on Claim 1, and claim 42 is dependent on claim 41. Claim 41 should be grouped with claim 42, but otherwise these claims do not fall together with claim 1, and should be considered separately and apart from all other claims. Claim 41 is based on a different ground of rejection based on the improper combination of WebWriter I and WebWriter II. Although claim 1 is formally rejected based on the combination of WebWriter I and WebWriter II, in that instance, WebWriter II is used in only for its recitation of server, while the examiner's reasoning for claim 41 relates more to WebWriter II and suggests a real combination of WebWriter I and II. In addition, Claims 41 and 42 require "*a client part for execution on the client computer*" while claims 1-6, 22, 26, 51, 74, 114 and 125 do not. Claim 59 and claim 90 have other specific features, as previously discussed, that make them separately patentable over claims 41 and 42. For these reasons, the group of claims 41 and 42 is separately patentable over all other claims discussed so far.

With respect to claims 41 and 42, the examiner cites WebWriter II at page 1510, section 4.1, as disclosing a client part for execution on the client computer. Applicant disagrees, and refers to section C.iii above for a discussion of the propriety of the combination of WebWriter I and WebWriter II. Specifically, claim 41 read together with the base claim 1 implicitly requires the editor client part to work on the documents generated by the document generator program on the server ("*an editor program dynamically operating on the generated documents*"). This feature is not shown by WebWriter I, which works totally on the server side and does not have a client part. It is also not shown by WebWriter II, which includes parts that work on the client side. Thus, the combination of these different implementations is not readily obvious because it would require a significant effort to provide even just a basic compatibility between these two versions of WebWriter.

For all these reasons, applicant submits that claim 41 should be considered patentable over the cited combination. Because Claim 42 is dependent on Claim 41, Claim 42 should be considered patentable over the cited combination for all the same reasons.

ii. Claim 8

Claim 8 is dependent on claim 6, but does not fall together with claim 6, and should be considered separately and apart from the other claims. This claim requires the important

distinctive feature of nested components as described in section B.ix. No other claims recite this feature except for claims 4, 81 and 87, and therefore, claim 4 cannot be grouped with any other claim except for claims 4, 81 and 87. However, since the base claim 6 requires the feature "*components have a similar appearance as on the generated page*," claim 8 cannot be grouped with any of these other claims.

With regard to claim 8, the examiner cited Fig. 4 of WebWriter I, just as with claim 4. Both claim 4 and claim 8 require that a component contain another component. Therefore, applicant submits that claim 8 is patentable over the cited combination because of the reasons discussed with regard to claim 4. Applicant also refers to the reasoning in section B.ix above showing that WebWriter does not teach or suggest nested components. In contrast, claim 8 explicitly requires that a component is nested within a component

iii. Claims 23-25

Claims 23-25 depend from Claim 22, and for all the same reasons, Claims 23-25 are not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. However, for the reasons discussed below, claims 23-25 do not fall together with claim 22, but must be considered separately.

a. Claim 23

Claim 23 should be considered separately and apart from the other claims. This claim requires the distinctive feature that "*the editor program operates a functional application in an edit mode*." No other claims recite this unique feature, and therefore claim 23 cannot be grouped with any claim not dependent on claim 23. In addition, claim 23 makes explicit that the edit mode allows "*editing of said application directly in the web browser*" which is also a feature not recited in any other claim.

With regard to Claim 23, the examiner cited page 2 column 1 of WebWriter I and contends that use of the term "template page" in the article means that the editor program operates functional applications in an edit mode. Claim 23 requires the edited application be functional during editing by claiming "*the editor program operates functional applications in an edit mode permitting editing directly in the web browser*". As reasoned in section A.i above, page 2, column 1 of the WebWriter I article does not teach that the edited application is running

during editing ("*the static parts of each page are created in advance by the designer using a text editor*"). Sections A.ii and C.i give several citations to the WebWriter articles that support the premise that, in fact, the application is run only after editing and not during editing. During editing, dynamic parts are displayed as placeholders (See page 4, col. 2, last para.: "*WebWriter then adds a placeholder for the output area"*). Also, operation in an edit mode as required by the claim is not taught or suggested in the cited prior art. For all these reasons, applicant submits that WebWriter does not teach or suggest "the editor program operates functional applications" as required by claim 23 and therefore, claim 23 is patentable over the cited combination.

The claim had been amended in previous responses to non-final office actions to recite "editing <u>of said application</u>" in order to emphasize that the term "application" as used in the claim indeed refers to the application that is being edited. The examiner, however, apparently did not take that amendment into account.

On page 28 of the office action dated July 1, 2009 the examiner refers to his argument for claim 90, which already has been discussed with claim 90.

b. <u>Claim 24</u>

Claim 24 depends from Claim 23 and for all the same reasons is patentable over the cited combination. However, Claim 24 does not fall together with Claim 23, and should be considered separately and apart from other claims. The claim includes the distinctive feature of components that react on subsequent document requests. No other claims recite this feature except for claims 3 and 78. Claim 24 cannot be grouped with claim 3 (or relevant dependent claims) because claim 24 depends on claim 23, which has features not recited or incorporated into claim 3. As further discussed below, none of the claims in claim set 74 including claim 78 can be grouped with other claim sets. Therefore, claim 24 cannot be grouped with other claims.

With regard to Claim 24, the examiner cited pp. 2-9 of WebWriter I as disclosing the claimed recitation of a component that can react on subsequent document requests by executing selected instructions. Applicant submits that the scripts called by WebWriter's dynamic areas are just used for processing of a single request. As explained in WebWriter I on page 9, left column, heading "The Web Writer Page Generator", the scripts are called during document generation to produce output for an output area during processing of a single request. The section does not discuss the concept of multiple requests or of subsequent requests and therefore does not teach

components that react in any way on a subsequent request. In contrast, claim 24 requires at least one component that can react on subsequent document requests. Please also refer to the discussion of this issue in section B.ix above. For all these reasons, applicant submits that claim 24 is patentable over the cited combination.

Further, on page 28 of the office action dated July 1, 2009, the examiner states that the plain language of the claims is directed to "subsequent document requests" and not to multiple requests. However, applicant merely notes that WebWriter's scripts work only during a single request and not during more than one i.e. multiple requests. A component able to handle subsequent requests, however, must be able to distinguish between an initial and a subsequent request. Under the Heading "The Web Writer Page Generator" the WebWriter I reference only describes the initial request that displays a dynamic area. It does not give any information on any subsequent requests, i.e. requests following the initial one.

c. Claim 25

Claim 25 depends from Claim 24, and for all the same reasons is patentable over the cited combination. However, Claim 25 does not fall with Claim 24 and should be considered separately and apart from the other claims because this claim requires the distinctive feature of a menu of components and the distinctive feature of a component class and because both of these features are not recited in any other claim.

For Claim 25, the examiner cited WebWriter I pp. 2-9, et seq. as disclosing component classes. However, applicant submits that neither WebWriter I nor WebWriter II discloses component classes implementing components. The scripts associated with dynamic areas are unix scripts or programs as disclosed on WebWriter I page 9, second column first paragraph ("*runs the named program ... The program is run in a separate Unix process*"). So the WebWriter Page Generator in fact calls unix scripts or programs. There is no teaching or suggestion in WebWriter that components are objects and implemented as classes.

In addition, WebWriter does not have a menu of components for insertion into the document templates. Figure 7(b) of the WebWriter I article shows forms for entering scripts, but not a menu.

Applicant submits that claim 25 is patentable over WebWriter I alone or in combination with WebWriter II since WebWriter does not teach or suggest component classes, a document generator program using component classes, nor a menu of components.

iv. Claims 27-33 and 43

Claims 27-33 and 43 depend from Claim 26, and for all the same reasons are patentable over the cited combination and cannot be grouped with a claim from any other claim set except 59, because claim 26 is the only claim besides 59 that requires the important distinctive feature that the second document appears and functions similar to the run-time view. Grouping with claim set 59 is discussed later.

a. Claim 27

Claim 27 should be considered separately and apart from the other claims. Unlike claim 26 this claim requires the important feature of a component generating code for the run-time view, and adds the distinctive requirement of components being executed by the first software program to the requirement for similar documents as recited in the base claim 26. Therefore it cannot be grouped with claim 27.

With regard to dependent claim 27, the examiner cites WebWriter I pages 2-9 et seq. as disclosing at least one component being executed by the first software program. Since the examiner simply cited almost the complete article he failed to give a detailed reasoning and so did not establish a *prima facie* case of obviousness.

Since base claim 26 requires the first software program to generate a second document having features which permit editing, the examiner must be interpreting the WebWriter Editor as the first software program. According to WebWriter I page 9 section entitled "The WebWriter Page Generator" (see also section B.ii above), it is the WebWriter Page Generator executing scripts associated with dynamic areas (see also Section B.ii) and not the WebWriter Editor. In contrast, the claim requires the first software program to execute at least one component. Therefore, applicant submits that claim 27 is patentable over the cited combination.

In addition the examiner is not taking all amendments of the claim into account. Since the functional parts of WebWriter documents are the scripts associated with dynamic areas and these are interpreted as components by the examiner, and claim 27 requires the first document to

include at least one component and claim 26 requires part of the second document to function similar to the runtime view, which in turn is required by claim 27 to have code being generated by the component this altogether requires a dynamic area in WebWriter that has similar functionality during editing and at runtime. This is, however, not the case in WebWriter since dynamic areas are shown as placeholders during editing as discussed in section A.ii. Therefore, applicant submits that claim 27 is patentable over the cited combination.

b. Claim 28

Claim 28 should be considered separately and apart from the other claims. It cannot be grouped with claim 26-27 because claim 28 requires the important distinctive feature of handles in a partly functional document ("*the second document includes handles*" whereby part of the second document is required to appear and function similar to the run-time view by base claim 26). In addition claim 28 is, besides claim 29 and claim 120, the only claim reciting handles. As discussed above claims from claim set 26 cannot be grouped with other claim sets, such as claim 120.

Claim 28 is dependent from claim 27 and claim 26, and for all the same reasons is patentable over the cited combination.With regard to claim 28, the examiner cites WebWriter I page 4 for "clicking on its handles" as disclosing that the second document includes handles and choosing one of the handles selects a component for an editing operation. According to the limitations of base claims , the second document is a document generated by the first software program (claim 26) by executing the component (claim 27). So, the claim language indeed requires handles to occur in a document generated by executing components. Also base claim 26 requires part of the second document to appear and function similar to the run-time view. In contrast, WebWriter shows handles during editing inside a document that contains placeholders and that does not function at all as described in section A.ii. In addition, the cited portion of the WebWriter I article seems to deal with HTML elements rather than dynamic areas, which are discussed later in the article. For all these reasons, applicant submits that claim 28 is patentable over the cited combination.

c. Claim 29

Claim 29 depends from claim 28 and for all the reasons given there cannot be considered together with any other claim. Claims 28 and 29 should be considered separately because claim 29 adds the limitation that the handle indicates the position of a component on the first document.

Claim 29 depends from Claim 28, and for all the same reasons is patentable over the cited combination. With regard to claim 29, applicant submits that the delete operation is not described in the cited portion of the WebWriter I reference.

d. Claim 30

Claim 30 should be considered separately and apart from the other claims. It cannot be grouped with claims 26-29 because base claim 26 requires the important feature of similarity of documents, and claim 30 in addition specifies scripts as a particular "feature" as used in claim 26. Also claim 30 and 31 are the only claims in claim set 26 reciting scripts. Claim 30 also cannot be grouped with claim 27 and its dependent claims because of the distinctive feature in claim 27 of "one component being executed". With regard to dependent claim 30, the examiner cites WebWriter I pages 2-9 et. seq. as disclosing that the features include scripts. The examiner did not cite a particular portion of the prior art that reveals the use of scripts inside the documents generated by the WebWriter I editor and so did not establish a *prima facie* case of obviousness. On page 8 right column second paragraph, the WebWriter Editor section discloses just HTML code. In contrast, the claim implicitly requires the second document to contain scripts, since claim 30 requires the features to include scripts and base claim 26 requires the second document to include these features. Therefore, applicant submits that claim 30 is patentable over the cited combination.

e. Claim 31

Claim 31 should be considered separately and apart from the other claims. Claim 31 cannot be grouped with claim 26-30 because unlike these claims claim 31 requires the important feature of scripts that encapsulate information of the first document which is not recited in any other claim.

Claim 31 depends from Claim 30, and for all the same reasons is patentable over the cited combination. With regard to dependent claim 31, the examiner cites WebWriter I pages 2-9 et. seq. as disclosing scripts that encapsulate information from the first document. However, the examiner did not cite a specific portion that reveals scripts being generated by the WebWriter Editor that encapsulate information from the first document. Therefore the examiner did not establish a *prima facie* case of obviousness and applicant submits that claim 31 is patentable over the cited combination.

f. Claim 33

Claim 33 should be considered separately and apart from the other claims. Claim 33 cannot be grouped with claim 26-32 because unlike these claims claim 33 requires the important feature of change requests, which is not recited in any other claim.

With regard to dependent claim 33, the examiner cites WebWriter II, page 1510 section 4.1 Applicant submits that a combination of WebWriter II and WebWriter I is improper as reasoned in section C.iii above, and for that reason, the citation of WebWriter II for the dependent claim is inadequate to teach or suggest the claimed invention.

In addition, the cited portion does not seem to reveal change requests. The WebWriter II seems to perform changes on the document tree and to send a complete document to the server after editing on a save operation. On page 1511, 3rd paragraph: "The CGI modules provide server-side services such as loading files, parsing HTML, **saving files** …" (emphasis added). In contrast to saving a complete document, claim 33 requires "*to send change requests for the first document to the server computer*", which mandates that a change be sent to the first document and not a complete document. For these reasons, applicant submits that claim 33 is patentable over the cited combination.

g. Claim 43

Claim 43 depends on claim 26. Claim 43 should be considered separately and apart from the other claims. It cannot be grouped with claim 26-33 because unlike these claims, claim 43 requires the important feature of a script for automatic download to the client that works in cooperation with the second document. Note that this feature is recited in claim 42 as well and similar reasoning applies, however, claim 43 cannot be grouped with claim 42 because claim 43

is dependent on claim 26, and claim 42 is dependent on claim 1 and because claim 26 includes important distinctive features as discussed above. In addition, claim 43 should be considered separately because the examiner applies a different ground of rejection.

With regard to dependent claim 43, the examiner cites WebWriter II, page 1510, section 4.1 Applicant submits that a combination of WebWriter II and WebWriter I is improper as reasoned in section C.iii, and for that reason, this claim is considered patentable over the cited combination.

v. Claims 53-58

Claims 53-58 depend from Claim 51, and for all the same reasons, these claims are patentable over the cited combination. Because only claim 51, claim 74, and claim 114 have the important distinctive feature of components cooperating with the editor, Claims 53-58 cannot be grouped with any claim not dependent on either 51,74, or 114. Grouping with these claims is discussed later.

a. Claim 53

Claim 53 should be considered separately and apart from all other claims. The claim stands rejected on different grounds than the other claims, except claims 33, 41-43, 60, 77 and 117, namely, the improper combination of WebWriter I and WebWriter II and can therefore not be grouped with any claim other than claims 33, 41-43, 60, 77 and 117. Although, e.g., claim 51 is rejected based on WebWriter II as well, it is used there only for its recitation of a server, while the examiner's reasoning for claim 53 references much more of the content of WebWriter II. However, this is an improper combination of WebWriter I and WebWriter II. Further, because of the cooperation feature recited in base claim 51, claim 53 can also not be grouped with claims 33, 41-43 and 60. In addition, claims 41-43 require that the editor program include a client part, while claim 53 just requires a **first** software program running on the client computer.

Claim 53 adds significant limitations stressing the client server nature of the system. With respect to claim 53, the examiner cited WebWriter II at page 1508, section 1.2, while he cited WebWriter I for the base claims 52 and 51. Applicant submits that for all the reasons given in section C.iii, the combination of WebWriter I and WebWriter II is improper.

b. Claim 54

Claim 54 should be considered separately and apart from the other claims. It cannot be grouped with claims 51-53 because, unlike these claims, claim 54 requires that the features to cooperate include instructions for collecting and passing information to the editor, and that second documents include HTML pages with embedded scripts.

With respect to claim 54, the examiner generally cites WebWriter I pages at 2-9 et. seq. as disclosing second documents that include HTML pages with embedded scripts. However, the examiner did not point to any particular portion of the prior art article that reveals the use of scripts inside the documents generated by the WebWriter I editor, and so, the Examiner did not establish a *prima facie* case of obviousness. On page 8, right column, second paragraph, the WebWriter Editor section discloses just HTML code. In contrast, the claim requires second documents to contain embedded scripts, and base claim 51 requires the second program to generate the second documents, and therefore applicant submits that claim 54 is patentable over the cited combination.

In addition, the examiner apparently did not take the recent amendment of the claim into account, and also did not cite a particular portion of the prior art that reveals that the features include instructions for collecting and passing information to the editor and instructions for displaying additional editing features during editing, and also for that reason did not establish a *prima facie* case of obviousness. Note that according to base claim 51, the features are claimed to be part of the components. The section labeled "WebWriter Page Generator" on page 9 of the WebWriter I article reveals that the scripts of dynamic areas are just general Unix scripts, and but does not discuss any special characteristics or instructions of these scripts, such as instructions to pass information to the editor. This would not be possible since according to the article, the scripts of dynamic areas are executed at run-time only as discussed in section B.ii. For all these reasons, applicant submits that claim 54 is patentable over the cited combination.

c. Claim 55

Claim 55 should be considered separately and apart from the other claims. It cannot be grouped with claim 51-54 because unlike these claims, claim 55 requires the feature of removing a component.

With regard to claim 55, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing edit functions include removing a component. However, the WebWriter I article does not appear to describe the function of deleting a component, and therefore the examiner's rejection is in error.

d. Claim 56

Claim 56 should be considered separately and apart from the other claims. It cannot be grouped with claim 51-55 because unlike these claims, claim 56 implicitly requires the distinctive feature of fourth program instructions of the components being called during editing.

As required by claim 51, the second software program makes the browser display a user interface for editing. This means that the second software program runs at editing-time. Claim 56 implicitly requires that fourth program instructions of components run at the same time, i.e. at editing-time. In contrast, scripts of dynamic areas are executed in WebWriter only at run-time, as discussed in section A.i and A.ii and B.ii. In addition, the examiner did not recite a specific portion of the prior art and so did not establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 56 is patentable over the cited combination.

Claims 57-58 depend from Claim 56, and for all the same reasons, these claims are patentable over the cited combination.

e. Claim 57

Claim 57 should be considered separately and apart from the other claims. It cannot be grouped with claims 51-56 because unlike these claims, claim 57 requires the important feature of generated documents to include edit features.

With respect to claim 57, the examiner cites WebWriter I at pages 2-9 and refers to the term "editor" as disclosing that the generated documents include edit features. The examiner did not recite a specific portion of the prior art and so has not established a *prima facie* case of obviousness.

The term "generated document" refers back to claim 56, where it is used as the documents generated by the WebWriter Page Generator and by executing instructions of the components. As reasoned in section C.i, however, the WebWriter Page Generator is separate from the WebWriter Editor and does not insert any editing features into the generated

documents. In contrast, the claim specifically requires that the generated documents include editing features. Therefore, applicant submits that claim 57 is patentable over the cited combination.

f. Claim 58

Claim 58 should be considered separately and apart from the other claims. It cannot be grouped with claims 51-57 because unlike these claims, claim 58 requires the important feature of instructions to allow the user to click on the generated documents for editing.

With respect to claim 58, the examiner cites WebWriter I at pages 2-9 and refers to the term "editor" as disclosing instructions to allow the user to click on the generated documents to select items to perform edit functions on. The term "generated documents," however, refers back to claim 56 , where it is used as the document generated by the WebWriter Page Generator and by executing instruction of the components. As reasoned in section C.i, however, the WebWriter Page Generator is separate from the WebWriter Editor. Therefore, it is not possible to click on a document generated by the WebWriter Page Generator to perform editing. In contrast, the claim requires instructions to allow the user to click on the generated document in order to select items to perform edit functions on. Therefore, applicant submits that claim 58 is patentable over the cited combination.

vi. Claims 60-73

Claims 60-73 are dependent from Claim 59, and for all the same reasons, applicant submits that Claims 60-73 are likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. Claim 59 has the limitation "dynamic web documents which look and function similar to the end user's view of the documents" which is not required by any other claim except Claim 26. In addition, claim 59 requires the editor to have instructions for requesting that the document generator process a dynamic document, which is not recited in any other claim. Therefore, only claims 59-73 have the combination of these features by virtue of dependency and therefore none of these claims can be grouped with any other claim.

a. Claim 60 and Claim 62

Claim 60 and dependent claim 62 form a group of claims that should be considered separately and apart from the other claims or groups. The claims cannot be grouped with base claim 59 because the rejection is based on a different ground of rejection, namely, the improper combination of WebWriter I and WebWriter II. Although independent claim 59 is formally rejected based on WebWriter II as well, it is used there only for its recitation of server, while the rejection of claim 60 relates much more to WebWriter II. Also, claim 60 introduces the limitation "the editor program at least partly running on the client computer". This feature makes the claim separately patentable over claim 59, and therefore, the group of claims 60 and 62 should be considered separately.

With regard to claim 60, the examiner cites section 4 of the WebWriter II article, for the document generator on the server, the editor at least partly on the client. However, the Examiner cited the WebWriter I article for base claim 59. Applicant submits that for all the reasons given in section C.iii above, the client side operations described in section 4 of WebWriter II cannot easily be combined with the server side functionality of WebWriter I. Therefore, the combination is improper and not effective to make the claims obvious.

Claims 61-63 depend from Claim 60, and for all the same reasons, these claims are patentable over the cited combination.

b. Claim 61

Claim 61 should be considered separately and apart from the other claims. This claim requires the important distinctive feature of instructions to collect edit-information during the document generation and therefore cannot be grouped with base claims 59-60.

With regard to claim 61, the examiner cited WebWriter I at pages 2-9 as disclosing the distinctive feature of instructions for execution **during document generation** to **collect edit information** for use by the editor. Since the examiner did not point to a specific portion of the prior art, and also because he did not consider and provide reasoning regarding the latest amendment of the claim, he has not established a *prima facie* case of obviousness.

In fact, the WebWriter I article describes two kinds of document generation, the document generation done by the WebWriter Editor on page 8, and the document generation done by the WebWriter Page Generator on page 9. According to base claim 59, "*the document*

generation" (note that recent amendments added "the") means "*generating generated documents from dynamic web documents which look and function similar to end user's view of the documents*". This is not the case for the document generation of the WebWriter Editor (see section A.ii) because it shows static placeholders for components that look different and do not function at all. For document generation of the WebWriter Page Generator, there is no teaching or suggestion of a feature "to collect edit-information" as required by the claim. Therefore, applicant submits that claim 61 is patentable over the cited combination.

c. Claim 63

Claim 63 should be considered separately and apart from the other claims. Claim 63 cannot be grouped with claim 59-62 because unlike these claims, claim 63 requires the important distinctive feature of instructions for automatically repeating the request.

With regard to Claim 63, the examiner cites WebWriter I at pages 2-9 for automatically repeating a request that the document generator process the dynamic web document if required. Since the examiner did not point to a specific portion of the prior art article, he did not establish a *prima facie* case of obviousness. As reasoned with regard to claim 59 and explained in sections A.i, A.ii, and C.i, the WebWriter Editor never does request that the WebWriter Page Generator generate a document, and so there cannot be any repeating of this step in WebWriter I. In fact, the WebWriter Page Generator is only running at run-time, when the WebWriter Editor has already saved the document. Therefore, applicant submits that claim 63 is patentable over the cited combination.

d. Claims 64 and 65

Claim 64 and dependent Claim 65 form a group that should be considered separately and apart from the other claims. Claim 64 requires components on document templates, and because of that, cannot be grouped with claim 59-63. Applicant refers to the reasoning for separate patentability of claim 1 with regard to components.

With regard to Claim 64, the examiner cites WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he has failed to establish a *prima facie* case of obviousness. The examiner specifically refers to the phrase "on the fly". The meaning of "on the fly", however, has already been discussed in detail in section C.i. Claim 64 requires

"*components ... including instructions for use by the document generator program to generate browser code*" and base claim 59 requires "*the editor program comprising first instructions for requesting that the document generator program processes a dynamic web document*". This means that components have to be used by a document generator that is requested by the editor. In contrast, the scripts of dynamic areas described in WebWriter are used only by the final document generator that runs after editing.

Claims 65 and 66 depend on claim 64, and for all the same reasons, should be considered patentable over the cited combination.

e. Claim 66

Claim 66 should be considered separately and apart from the other claims. Claim 66 cannot be grouped with claim 59-65 because this claim requires the important distinctive feature of deleting a component.

With regard to Claim 66, the examiner cites pages 2-9 of the WebWriter I article as disclosing the operations of inserting, deleting and modifying a component. However, the WebWriter I article does not describe the deletion of a component.

f. Claim 67

Claim 67 should be considered separately and apart from the other claims. Claim 67 cannot be grouped with claim 59-67 because this claim requires the important distinctive feature of the view looking, except for editing features, similar to the end-user view.

With regard to Claim 67, the examiner cites pages 2-9 of WebWriter I as disclosing that the view looks, except for editing features, similar to the end-user view. In general, both views do not look similar, as explained in section A.ii above. In fact, Figs. 7(b) and 6(b) of WebWriter I show both views, which clearly look different since the end user's view contains the output generated by scripts associated with dynamic areas while the view shown by the editor just shows placeholders. Also, see Figs. 10(b) and 11 for another example. In contrast, the claim requires both views to look similar except for editing features. Therefore, applicant submits that claim 67 is patentable over the cited combination. Also, since the examiner did not recite a specific portion of the prior art, he has failed to establish a *prima facie* case of obviousness.

g. Claim 68

Claim 68 should be considered separately and apart from the other claims. Claim 68 requires sixth instructions to collect edit-information for execution during the document generation. Because claims 59 -60 and 62-67 do not require collection of edit information, claim 68 cannot be grouped with any of these claims. Claim 61 requires collection of edit information, however, it is dependent on claim 60, which has another ground of rejection and additional features, and therefore cannot be grouped with claim 68. Nevertheless, applicant submits that the reasoning for claim 61 applies for claim 68 as well, and that claim 68 is patentable over the cited combination. Claim 68 was also amended in the same way as claim 61, but it appears that the amendment was not considered by the examiner.

h. Claim 69

Claim 69 should be considered separately and apart from the other claims. Claim 69 cannot be grouped with claim 59-68 because, unlike these claims, claim 69 requires the important distinctive feature of using the edit-information to correctly modify the dynamic web document.

Claim 69 is dependent on 68, and for all the same reasons, applicant submits that Claim 69 is likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. With regard to claim 69, the examiner cited WebWriter I at pp. 2-9 as disclosing the editor using the edit information to modify the dynamic web document. However, in WebWriter I, the edit information is never collected as explained above with respect to claim 61. Therefore, applicant submits that claim 69 is patentable over the cited combination. Also, the examiner did not recite a specific portion of the prior art, and therefore he failed to establish a *prima facie* case of obviousness.

i. Claim 70

Claim 70 should be considered separately and apart from the other claims. It cannot be grouped with claims 59-69 because unlike these claims, claim 70 requires the important distinctive feature of position information on selected components marked on the dynamic web document.

Claim 70 is dependent on 69, and for all the same reasons, applicant submits that Claim 70 is likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. With regard to claim 70, the examiner cites WebWriter I at pp. page 2-9. In the office action dated July 1, 2009, on page 31, the examiner cites under the heading Web Writer editor, that "... *Web Writer inserts a new object* of *the requested class into the content tree data structure at the position designated by the insertion point and sets this new object to be the selected object... ".* In applicant's reading, this discloses position information in the document tree and not position information on the document template. In contrast, Claim70 requires "*position information on the components contained in the document template*". Applicant therefore submits that claim 70 is patentable over the cited combination.

j. Claim 72

Claim 72 should be considered separately and apart from the other claims. It cannot be grouped with claim 59-71 because unlike these claims, claim 72 requires the important distinctive feature of instructions for initiating a reload.

Claim 72 is dependent on claim 71, and for all the same reasons, applicant submits that Claim 70 is likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. With regard to claim 72, the examiner cited WebWriter I at pp. 2-9 as disclosing initiating a reload in the browser. In the office action dated July 1, 2009, on page 31, the examiner cites a portion on page 8 just on top of the headline "Discussion of the WebWriter Approach" reciting refresh meta tags. The cited portion, however, discusses the Walkie-talkie sample application and proposes to add the refresh meta tag to that application, and not to the editor itself. Claim 72, however, requires the reload to be performed by first instructions, which are part of the editor as claimed in the base claim 59. In addition, the refresh meta tag technically does a redirect to another page rather than reloading the current one. Even if the meta tag is used to redirect to the same page, such an operation loses possible URL or form parameters. In contrast, claim 72 explicitly requires a reload in the browser. The examiner also cites an operation of loading a stack. This operation is clearly different from the well defined reload function of a web browser.

A reload in the browser means a function of the browser to load the same page a **second time**. However, WebWriter I does not describe the editor as containing instructions for initiating

a reload, as required in Claim 72. Therefore, applicant submits that claim 72 is patentable over the cited combination.

k. Claim 73

Claim 73 should be considered separately and apart from the other claims. It cannot be grouped with claims 59-72 because unlike these claims, claim 73 requires the important distinctive feature of displaying without requesting that the document generator generates a document.

With regard to claim 73, the examiner generally cites WebWriter I at pp. 2-9. Since the examiner did not recite a specific portion of the prior art article, he has failed to establish a *prima facie* case of obviousness. Claim 73 requires information on an element to be displayed without requesting that the document generator generate a document. In contrast, WebWriter I **does** have a page generation part in the WebWriter Editor, when showing information on a dynamic area and associated script (as described below). In that situation, there is no page generation in the WebWriter Page Generator since the WebWriter Page Generator runs only at run-time after editing. However, as explained in the reasoning of claim 59 and sections A.i, A.ii and C.i, the WebWriter Page Generator is not a page generator in the sense of base claim 59 and therefore not relevant for the claim. In order to display information on an element, the user must click a handle. The 2nd paragraph on page 9, left column, for WebWriter I states: "The next time WebWriter is called (after the user clicks on a handle or control panel button)," which makes clear that clicking a handle makes WebWriter **request** the WebWriter Editor to generate a document. Applicant's editor, in contrast, does have a client part that can process selected mouse clicks on handles, without requesting that the server part generate a page (and consequently without delay of a server interaction). So, because WebWriter I requires a page generation to display information on an element, but claim 73 requires this operation to be done without performing a request, the claim is patentable over WebWriter I.

vii. Claims 75-89

Claims 75-89 depend from claim 74, and for all the same reasons, should be considered patentable over the cited combination. Because claim 74 requires "cooperate with the editor" while claim 5 does not, claim 75-89 cannot be grouped with claim 5. Further, since only claim 74

and claim 5 have the important distinctive feature of a varying component set, Claims 75-89 cannot be grouped with any claim not dependent on claim 74.

a. Claim 75

Claim 75 should be considered separately and apart from the other claims. It cannot be grouped with claim 74 because this claim additionally requires the feature of deleting a component.

With regard to claim 75, the examiner cites WebWriter I at pp. 2-9 and refers use of to the term "editor" as disclosing editing functions comprising adding, modifying and deleting a component. However, the function of deleting a component does not seem to be described in the cited document.

b. Claim 76

Claim 76 should be considered separately and apart from the other claims. It cannot be grouped with claims 74-75 because unlike these claims, claim 76 requires the important distinctive feature of a specific tag syntax.

With regard to claim 76, the examiner cited WebWriter I at pp. 2-9 and refers to use of the term "templates" as disclosing components denoted on document templates using tag syntax, whereby the tag name identifies the component kind. However, WebWriter I on page 9, left column, discloses that output areas with scripts are denoted using a tag of the form:

<output script="…">formatting string</output>

The tag name of this syntax is "output" while it is the script attribute that identifies what the script is to be called. In contrast, however, claim 76 explicitly requires the tag name to identify the component kind. Therefore, applicant submits that claim 76 is patentable over the cited combination.

c. Claim 77

Claim 77 should be considered separately and apart from the other claims. It cannot be grouped with claims 74-76 because the rejection of claim 77 is based on a different ground of rejection, namely, the improper combination of WebWriter I and WebWriter II. In addition,

claim 77, unlike claims 74-76, requires the distinctive feature of an editor program at least partly running on the client computer.

With regard to claim 77, the examiner refers to a combination of WebWriter I and WebWriter II for document generator on the server and the editor at least partly on the client.. Applicant refers to section C.iii and submits that a combination of WebWriter I and WebWriter II is improper, and should be withdrawn as a basis for rejection.

In the office action dated July 1, 2009, on pages 31-32, the examiner argues "Web Writer II essentially offers an improvement over the initial Web Writer version". Applicant strongly disagrees. Although one might assume that from the naming of WebWriter and WebWriter II, WebWriter II is not a newer improved version of WebWriter. WebWriter II is a totally different approach for constructing an editor. WebWriter II has an implementation architecture (called "Meteor shower") which is totally different and incompatible with WebWriter I. It therefore has other advantages and disadvantages relative to WebWriter I. While WebWriter I is essentially a C++ program running on a server, WebWriter II is a javascript program running on a client. (*See* WebWriter II, page 1508, second paragraph). Moreover, the new Meteor Shower architecture teaches away from running during editing, since the editor essentially operates on the client and transfers the edited application to the server on a save operation. In contrast, execution of the edited server side web application (either during editing or not) requires that the application be on server side. See section C.iii for details.

d. Claim 78

Claim 78 should be considered separately and apart from the other claims. Claim 78 cannot be grouped with claims 74-77 because unlike these claims, claim 78 requires the important feature of components reacting interactively on subsequent requests.

With regard to claim 78, the examiner cites WebWriter I at pp. 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to use of the terms "generator" and "templates" as disclosing that at least one component can be excluded from the generated document. However, the section "The Web Writer Page Generator" on page 9 makes clear that the WebWriter Page Generator processes each dynamic area on a template page by executing the associated script and has no ability to exclude a dynamic area from this process. In contrast, the claim requires "*at least one*

component that ... can be excluded from said generated document upon selected document requests for said document template".

In addition, the claim requires a component that can react interactively on subsequent document requests. As discussed in section B.ix, WebWriter does not seem to disclose interactive components. For all these reasons, applicant submits that claim 78 is patentable over the cited combination.

Claims 79-80 depend from Claim 78, and for all the same reasons, these claims are patentable over the cited combination.

e. Claim 79

Claim 79 should be considered separately and apart from the other claims. Claim 79 cannot be grouped with claims 74-78 because unlike these claims, claim 79 requires the important distinctive feature of instructions to prevent excluded components from reacting on subsequent document requests.

With regard to claim 79, the examiner cites WebWriter I at pages 2-9 as disclosing instructions to prevent excluded components from reacting on subsequent document requests. As discussed in section B.ix, scripts of dynamic areas of WebWriter cannot react on subsequent document requests and therefore WebWriter also does not include instructions to prevent this reacting. Also, since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 79 is patentable over the cited combination.

f. Claim 80

Claim 80 should be considered separately and apart from the other claims. Claim 80 cannot be grouped with claims 74-79 because unlike these claims, claim 80 requires the important distinctive feature of storing information in session memory on some of the components.

Claim 80 depends from Claim 79, and for all the same reasons, this claim is patentable over the cited combination.With regard to claim 80, the examiner cites WebWriter I at pages 2-9 as disclosing storing information in session memory on some components that are present on the document generated. However, the section entitled "The Web Writer Page Generator" on page 9

does not refer to session memory or storing of information in session memory on scripts of dynamic areas. The claim refers to actions taking place at run-time and therefore would have been described in the cited section if they had been present in WebWriter I. Thus, applicant submits that there is no teaching or suggestion for storing of information on scripts of dynamic areas in session memory in WebWriter and therefore the claim should be patentable over WebWriter.

In the office action dated July 1, 2009, on page 32, the examiner refers to "saving and restoring the state of runtime data structures." in the related work section. Claim 80, however, contains limitations on what information to store in session memory, namely, "store information in session memory on some of the components that are present on the document generated" and also limitations on how that information is used later on, namely, "only react on components that have been remembered in session memory". The recited portion of the prior art does not teach or disclose any of these two limitations. For all these reasons, applicant submits that claim 80 is patentable over the cited combination.

g. Claim 81

Claim 81 should be considered separately and apart from the other claims. Claim 81 cannot be grouped with claims 74-80 because unlike these claims, claim 81 requires the important distinctive feature of instructions to decide upon exclusion of a component.

With regard to claim 81, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "template" as disclosing a first component containing instructions to decide about **exclusion** of components nested inside the first component. However, the section "The Web Writer Page Generator" on page 9 makes clear that the WebWriter Page Generator processes each dynamic area on a template page by executing the associated script and has no ability to exclude a script of a dynamic area from this process. In the office action dated July 1, 2009, on page 32, the examiner argues that he believes WebWriter teaches instructions to decide about exclusions of components. He says: "...where each page can contain both static content that is always present and computed content that is generated on the fly ... ". In applicant's reading, this portion does not reveal an operation of excluding a complete dynamic area from a page. It just reveals that the content of dynamic areas is computed at run-time. In contrast, the section entitled "The Web Writer Page Generator" on page 9 makes clear that the WebWriter page generator

processes each dynamic area on a template page by executing the associated script and has no ability to exclude a script of a dynamic area from this process. In contrast, the claim requires "*first component contains sixth instructions to decide about exclusion of components nested inside the first component*" and therefore claim 81 is patentable over the cited combination.

In addition, claim 81 requires that components be nested inside the first component. Applicant refers to section B.ix for the discussion that WebWriter does not disclose nested components.

In the office action dated July 1, 2009, at the bottom of page 32, the examiner argues that components on a document template are an example of nested components. Claim 74, however, lists component and document templates as different items and requires components denoted on document templates. Since document templates are not denoted on document templates in WebWriter, document templates are not components in the sense of the claim. Therefore, a dynamic area on a document template cannot be interpreted as prior art for nested components. Consequently, the examiner's argument is wrong, and the examiner has not identified any relevant prior art for the nested components. In contrast, claim 81 requires "*components nested inside the first component*" and also for this reason claim 81 is patentable over the cited combination.

h. Claim 82

Claim 82 should be considered separately and apart from the other claims. Claim 82 cannot be grouped with claims 74-81 because unlike these claims, claim 82 requires the important distinctive feature of an editable view taking the varying set of components into account.

With regard to claim 82, the examiner cites WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to the term "display/interface" as disclosing an editor taking the varying set of components into account. However, the WebWriter Editor displays each dynamic area with associated script as exactly one placeholder. There is no mechanism in the WebWriter Editor to take a varying set of components into account. In contrast, the claim requires the editor be able to take the varying set of components into account. Also, as discussed with regard to claim 74, the WebWriter Page Generator cannot handle a varying set of

components. For all these reasons, applicant submits that claim 82 is patentable over the cited combination.

i. Claim 83

Claim 83 should be considered separately and apart from the other claims. Claim 83 cannot be grouped with claims 74-82 because unlike these claims, claim 83 requires the important distinctive feature of an editable view that includes and excludes components.

With regard to claim 83, the examiner cited WebWriter I at pages 2-9 and refers to the terms "editor" and "template" as disclosing an editable view that includes and excludes selected components. However, the WebWriter Editor displays each dynamic area with associated script as exactly one placeholder. There does not seem to be a view that includes and excludes placeholders. In contrast, claim 83 requires a view that includes and excludes selected components. Also, the examiner did not recite a specific portion of the prior art, and therefore he did not establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 83 is patentable over the cited combination.

j. Claim 84

Claim 84 should be considered separately and apart from the other claims. Claim 84 cannot be grouped with claims 74-83 because unlike these claims, claim 84 requires the important feature of a generated document containing more components than its document template. Note that this is different from the dynamic set of components as required by claim 74, since a dynamic set could just mean a smaller number e.g. by exclusion of components.

With regard to claim 84, the examiner cites WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to the term "template" as disclosing a document generated for at least one document template that contains more components than the document template. In WebWriter, each dynamic area is filled by its script exactly once, as explained in the section entitled "The Web Writer Page Generator" on page 9. Therefore, a page generated by the WebWriter Page Generator cannot contain more components than the template page. In contrast, however, the claim requires that a generated document contains more components that the

document template. For all these reasons applicant submits that claim 84 is patentable over the cited combination.

k. Claim 85

Claim 85 should be considered separately and apart from the other claims. Claim 85 cannot be grouped with claims 74-84 because unlike these claims, claim 85 requires the important distinctive feature of multiple instances of a component.

With regard to claim 85, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "templates" as disclosing multiple instances of a component being included in a generated document. In WebWriter, each dynamic area is filled by its script exactly once, as explained in the section entitled "The Web Writer Page Generator" on page 9. In contrast, claim 85 requires multiple instances. Also, the examiner did not recite a specific portion of the prior art, and therefore he did not establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 85 is patentable over the cited combination.

l. Claim 86

Claim 86 should be considered separately and apart from the other claims. Claim 86 cannot be grouped with claims 74-85 because unlike these claims, claim 86 requires the important distinctive feature of assigning unique identifiers and qualifying names.

With regard to claim 86, the examiner cited WebWriter I under the heading The Web Writer Application Model "a set of named form elements used as variables" as disclosing instructions to assign unique identifiers and to qualify names generated into the browser code with the unique identifiers. The cited portion, however, just seems to reveal a set of previously named form elements. This is possible in WebWriter since each dynamic area is contained exactly once on the generated page. In contrast to previously named elements as disclosed by WebWriter, the claim requires **instructions** to generate names, more precisely instructions to assign unique identifiers and instructions to qualify names. Note that the claimed instructions are needed for multiple instances of components to ensure that identifiers generated into the browser code stay unique where required.

Thus, applicant submits that claim 86 is patentable over the cited combination.

m. Claim 87

Claim 87 should be considered separately and apart from the other claims. Claim 86 cannot be grouped with claims 74-86 because unlike these claims, claim 87 requires the important distinctive feature of instructions to decide how many instances of component are included.

With regard to claim 87, the examiner cites WebWriter I under the heading The Web Writer Application Model "one or more dynamic areas on each template page" as disclosing instructions to decide how many instances of components are included in the documents generated.

In applicants reading, the cited portion just reveals one or more dynamic areas on a template document. The claim, however, refers to how often a single component on the document template is included in the generated document. The use of the term "instance" should make this clear. In addition, the term "component instance" is clearly defined in applicant's specification. In contrast, WebWriter fills each dynamic area by its script exactly once, as explained in the section entitled "The Web Writer Page Generator" on page 9, so the cited article does not disclose multiple instances.

In addition, the cited portion "one or more dynamic areas on each template page" just states that the user may put one or more dynamic areas on each template, it does not reveal any instructions to decide about the number of instances. In contrast, the claim requires instructions to decide how many instances of components are included in the documents generated. The examiner also adds "Examiner interprets this to be nested components". Applicant already discussed with claim 81 that a document template is not a component and therefore a dynamic area on a template page is not an example for a component inside another one.

For all these reasons, applicant submits that claim 87 is patentable over the cited combination.

n. Claim 88

Claim 88 should be considered separately and apart from the other claims. Claim 88 cannot be grouped with claims 74-87 because unlike these claims, claim 88 requires the important distinctive feature of an editable view that includes multiple instances of components.

With regard to claim 88, the examiner cites the WebWriter I section under the heading "Script and Formatting of Output areas" and states that multiple instances of variables are disclosed. In applicants reading of the cited portion and of the section entitled "The Web Writer Page Generator" on page 9, each variable on the template document is displayed exactly once in the end user's view of the document template, so there are no multiple instances of variables in WebWriter I.

In addition, variables are not components in the sense of base claim 74, because claim 74 requires components having instructions to generate browser code. Variables in WebWriter do not have any instructions. Therefore, the examiner's discussion of multiple instances of variables is irrelevant, because claim 88 requires multiple instances of selected components.

In the office action dated July 1, 2009, on page 33, the examiner refers to the reasoning applied to claim 81. Claim 88, however, deals with a view of multiple instances of a component while claim 81 with excluding components.

As reasoned with regard to claim 85, WebWriter does not teach or suggest multiple instances of components at run-time and therefore it is clear that it does not have multiple instances in an editable view as well. Also, the description of the WebWriter Editor does not reveal multiple instances of placeholders or of scripts of dynamic areas. In contrast, the claim explicitly requires an editable view that includes multiple instances of selected components. Therefore, applicant submits that claim 88 is patentable over the cited combination.

o. Claim 89

Claim 89 should be considered separately and apart from the other claims. Claim 89 cannot be grouped with claims 74-88 because unlike these claims, claim 89 requires the important distinctive feature of tenth instructions displaying a component during editing as well as during normal use.

With regard to claim 89, the examiner cites WebWriter I at pp. 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to use of the term "browser" as disclosing a sixth component including tenth instructions for displaying the sixth component during editing. Applicant submits that as reasoned in section A.ii, the WebWriter Editor shows placeholders during editing and does not execute any instructions of the scripts associated with dynamic areas during editing.

Therefore, applicant submits that WebWriter does not disclose tenth instructions. In contrast, the claim requires a sixth component including tenth instructions for displaying the sixth component during editing. Therefore, applicant submits that claim 89 is patentable over the cited combination.

viii. Claims 91-96

Claims 91-96 depend from Claim 90, and for all the same reasons are patentable over the cited combination. Because claim 90 requires "*scripts remain running during editing*" and "*a first software program for execution within the browser*" and because claim 90 is the only claim with this feature combination, Claims 91-96 cannot be grouped with any claim not dependent on claim 90.

a. Claim 91

Claim 91 should be considered separately and apart from the other claims. Claim 91 cannot be grouped with claim 90 or any other claim because this claim additionally requires the important distinctive feature of at least two windows.

With regard to claim 91, the examiner cited WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. In applicants reading, WebWriter I just uses a first browser window, for example, as shown in figure 1. In contrast, claim 91 explicitly requires a "second window". WebWriter II appears to use frames and not a second window. The examiner's argument in office action dated July 1, 2009 on page 33 is no longer applicable because applicant's argument has been corrected to argue about a "second window" rather than a "second browser window". For all these reasons, applicant submits that claim 91 is patentable over the cited combination.

Claims 93-95 depend from Claim 92, and for all the same reasons are patentable over the cited combination.

b. Claim 93

Claim 93 should be considered separately and apart from the other claims. Claim 93 cannot be grouped with claims 90-92 because unlike these claims, claim 93 requires the important distinctive feature of said view looking similar to the original document.

With regard to Claim 93, the examiner cites WebWriter I at pages 2-9 as disclosing that the original document looks similar to the document. Applicant submits that in general the original document and the document shown during editing look clearly different in WebWriter since dynamic areas are replaced by placeholders. This is discussed in section A.ii. Also, since the examiner did not recite a specific portion of the prior art, he has failed to establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 93 is patentable over the cited combination.

Claims 94-95 depend from Claim 93, and for all the same reasons are patentable over the cited combination.

c. Claim 94

Claim 94 and 95 form a group that should be considered separately and apart from the other claims. Claim 94-95 cannot be grouped with claims 90-93 because unlike these claims, claim 94 requires the important feature of the original document having components.

With regard to Claim 94, the examiner cites WebWriter I at pages 2-9 and refers to the term browser. The claim explicitly requires the original document to be a dynamic document containing components with instructions contained therein. As explained in section A.ii, in the case of dynamic documents, the WebWriter Editor displays a document that appears different than the original document since dynamic areas are replaced by placeholders. In contrast, base claim 93 requires that the original document and the document look similar. Also, since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 94 is patentable over the cited combination.

d. Claim 95

Claim 95 depends from Claim 93, and for all the same reasons is patentable over the cited combination.

e. Claim 96

Claim 96 should be considered separately and apart from the other claims. Claim 96 cannot be grouped with claims 90-95 or any other claim because unlike these claims, claim 96 requires the important feature of links staying functional during editing.

With regard to claim 96, the examiner cites WebWriter I at pages 2-9 as disclosing that links contained in the document stay functional during editing. The cited portion does not seem to include any teaching on how links are displayed in a document being edited. Applicant further submits that simply including a link in a document edited by the WebWriter Editor would leave the editor dysfunctional, because WebWriter stores all its internal data structures into a hidden field as described in the section called "The WebWriter Editor" on page 9. It is the standard behavior of a browser to send the content of the hidden field to the server, only when a button is pressed to submit the form. When a link is pressed, browsers lose the content of the hidden field, and therefore WebWriter would lose its internal data structures in that case. Therefore, applicant assumes that links in the edited document template are disabled in the WebWriter editor. In contrast, applicant's claim requires that links stay functional during editing in order to allow the user to browse and edit at the same time. Also, since the examiner did not recite a specific portion of the prior art, he failed to establish a *prima facie* case of obviousness. For all these reasons, applicant submits that claim 96 is patentable over the cited combination.

ix. Claims 115-124

Claims 115-124 and 128 depend from claim 114, and for all the same reasons, should be considered patentable over the cited combination. Because claim 114 requires the important distinctive feature of components having features to cooperate with the editor, claims 115-124 could possibly be grouped only with claims that have the same feature, i.e. claims dependent on claim 51, claim 74, or claim 114. Claim set 74, however, as has been discussed, cannot be grouped with any other claims. Since claim 51 requires additionally "a second software program transmitting, while processing selected requests, second documents to the first software program that make the first software program display a user interface for editing functions used for maintaining components on document templates" all claims dependent on claim 51 have the

same by dependency. Since none of Claims 115-124 and 128 share a similar feature, Claims 115-124 and 128 cannot be grouped with any of the claim set 51.

a. Claim 115

Claim 115 should be considered separately and apart from the other claims. It cannot be grouped with Claim 114 or Claim 128 because unlike these claims, claim 115 requires the distinctive feature of components having fourth instructions for passing information to the editor. It cannot be grouped with a claim from any other claim set either as discussed above.

With regard to claim 115, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing that first features include instructions for passing information to the editor. According to base claim 114, components for execution on the server include first features for cooperation with the editor. Claim 115 states "*first features include fourth program instructions for passing information to the editor*" and depends on claim 114, which states that "*at least one of the components including first features.*" WebWriter does not describe that scripts called by WebWriter's dynamic areas pass information to the editor. In fact, as discussed in section A.ii and section C.i, these scripts are executed at run-time, after editing is finished, and so the scripts running later cannot pass information to the editor as required by the claim. Consequently, WebWriter's scripts called by dynamic areas have no features for passing information to the editor. In contrast, claim 115 requires these features and therefore applicant submits that claim 115 should be patentable over the cited combination.

In the office action dated July 1, 2009, on page 35, the examiner refers to "dynamic areas of editor as disclosed by Web Writer on page 2". In applicant's reading of this portion, no "*instructions for passing information to the editor"* are disclosed. In applicant's reading, page 2, left column under the headline *"The WebWriter Application Model"* last paragraph *"specification of locations within the page, called dynamic areas, where computed content should be inserted at* ***runtime*** *to produce the final page that should be displayed to the user"* (emphasis added) just discloses instructions "*to produce the final page that should be displayed to the user"* but no instructions to pass information to the editor. In addition, these instructions are used at runtime after editing and therefore cannot pass information to the editor. Because the cited portion does not disclose instructions for passing information to the editor, examiners

argument is flawed. For all these reasons, applicant submits that claim 115 is patentable over the cited combination.

Claims 116-118 depend from Claim 115, and for all the same reasons, these claims are patentable over the cited combination.

b. Claim 116

Claim 116 should be considered separately and apart from the other claims. It cannot be grouped with Claim 114-115 or claim 128 because unlike these claims, claim 116 requires the important distinctive feature of collecting information during execution of components.

With regard to claim 116, the examiner cited WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to the cited portion as disclosing that part of the information is collected during execution of the components on the server. However, the scripts called by WebWriter's dynamic areas do not collect information for the editor. As discussed in section B.ii, A.ii and C.i, these scripts run at run-time after editing is finished and therefore cannot pass information to the editor. In contrast, claim 116 requires that part of the information is collected during execution of the components. Therefore, applicant submits that claim 116 should be patentable over the cited combination.

c. Claim 117

Claim 117 should be considered separately and apart from the other claims. Claim 117 requires, together with its base claims, the unique distinctive feature of "components including first features including fourth program instructions for passing information to the editor" and "said information is transmitted from the server computer to the client computer". Because no other claim has this unique distinctive feature, this claim should be considered separately.

Also, Claim 117 should be considered separately and apart from the other claims because the rejection is based on the improper combination of WebWriter I and WebWriter II as discussed in section C.iii.

With regard to claim 117, the examiner refers to WebWriter II. Applicant submits, however, that "said information" is in fact never collected as reasoned with regard to base claim 115, and therefore also not transmitted either by WebWriter I or by WebWriter II.

In addition, the examiner did not recite a specific portion in the cited art describing transmission of information of a component being edited to the client computer, and so failed to establish a prima face case of obviousness. In fact, scripts associated with dynamic areas, as described in WebWriter I and discussed in section B.ii, run after editing and therefore no information can be transmitted to the editor.

d. Claim 118

Claim 118 should be considered separately and apart from the other claims. Because this claim requires the important feature of said information includes attribute values of a component and because claims 114-117 and claim 128 do not have this feature, claim 118 cannot be grouped with these claims.

With regard to claim 118, the examiner cited WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to use of the term "editor" as disclosing that the information includes attributes of the component. As reasoned with regard to base claims 114 and 115, no information is transmitted from the running scripts associated with WebWriter's dynamic areas to the WebWriter Editor. In addition, the cited portion does not show that the scripts called by WebWriter's dynamic areas collect any attribute values to be passed to the editor. In contrast, base claims 114 and 115 require components to include fourth program instruction for passing information to the editor, and claim 118 requires the information to include attribute values of components. For these reasons, applicant submits that claim 118 is patentable over the cited combination.

e. Claim 119

Claim 119 should be considered separately and apart from the other claims. Because this claim requires, unlike claims 114-118 and claim 128, the important distinctive feature of "first features include instructions to display additional editing features," it cannot be grouped with these claims.

With regard to claim 119, the examiner cited WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to the use of term "editor" as disclosing first features

including fifth instructions that display additional editing features. The base claim 114 requires "*components including first features.*" In applicants reading, the cited portion does not reveal that scripts called by WebWriter's dynamic areas display any additional editing features. In fact, as reasoned in section A.i, A.ii and C.i, these scripts run at run-time only, i.e., after editing. Therefore, it makes no sense for them to display any editing features since editing is already finished. In contrast, claim 119 together with base claim 114 requires components including first features including instructions that display additional editing features. For these reasons, applicant submits that claim 119 is patentable over the cited combination.

f. Claim 120

Claim 120 should be considered separately and apart from the other claims. Claim 120 cannot be grouped with claims 114-119 and claim 128 because, unlike these claims, claim 120 requires the important distinctive feature of said editing features include handles.

Claim 120 depends from Claim 119, and for all the same reasons, this claim is patentable over the cited combination. With regard to claim 120, the examiner cites WebWriter I at pages 2-9 . Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to use of the term editor as disclosing that said editing features include handles. In applicants reading, handles are displayed by the WebWriter Editor. In contrast, claim 120 requires said editing features to include handles, base claim 119 requires first features include instructions that display additional editing features, and base claim 114 requires components including first features. So, the claim requires **components** to contain instructions to display handles. In contrast, in WebWriter it is the editor displaying the handles. For these reasons, applicant submits that claim 119 is patentable over the cited combination.

g. Claim 121

Claim 121 should be considered separately and apart from the other claims. Claim 121 cannot be grouped with claim 114-120 and claim 128 because unlike these claims, claim 121 requires the important distinctive feature of an extension for enabling editing of attribute values of components.

With regard to claim 121, the examiner cited WebWriter I at pages 2-9 and refers to the term editor as disclosing first features including extensions for use by the editor. Claim 121 recites "*first features include an extension for use by the editor"* and read together with the base claim 114 "*at least one of the components including first features*" limits the extensions to be part of the components and not of the editor. The cited portion of WebWriter does not reveal any extensions for use by the editor being contained in the scripts called by WebWriter's dynamic areas. The scripts are used only at run time, when the editing is finished, as reasoned within sections B.ii, A.ii and C.i. Therefore, applicant submits that WebWriter does not teach extensions as required by the claim.

In the office action dated July 1, 2009, on page 34, the examiner introduces various elements including HTML tags and interprets these as applicant's term extension. The claim, however, requires an "extension for enabling editing of an attribute value of the components". The items cited by the examiner appear unrelated to editing of attribute values of dynamic areas and therefore are not an extension as required by claim 121. In addition, the portion cited in the action on page 34 reads "Web Writer can show HTML tags … ", which makes clear that the portion talks about items **part of the WebWriter editor**. In contrast, claim 114 and claim 121 require that the extension be a **part of the components**: "*components including first features*", "*first features include an extension*."

WebWriter has a single creation form for output areas which is built into the WebWriter Editor as disclosed on page 5 under the headline Scripts "*The user specifies a script by filling in **the** creation form for an output area. The user selects from a small set of **built-in** modules ...*" (emphasis added). In contrast, claim 121 requires first features include an extension for use by the editor for enabling editing of an attribute value of the components, and claim 114 requires these first features being part of the components. In contrast with WebWriter, the creation form is built into the WebWriter editor directly.

For these reasons, applicant submits that claim 121 is patentable over the cited combination.

h. Claim 122

Claim 122 should be considered separately and apart from the other claims. Claim 122 cannot be grouped with claims 114-121 and claim 128 because unlike these claims, claim 122 requires the important distinctive feature of a page for editing the attribute values of components.

Claim 122 depends from Claim 121, and for all the same reasons, this claim is patentable over the cited combination. With regard to claim 122, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing a web page for editing component attributes. Claim 122 explicitly requires "*said extension enables display of a page for editing the components attributes values.*" As reasoned with regard to base claim 121, the base claims require the extension to be part of components. In contrast, all the forms shown by the WebWriter Editor are built into the editor itself.

For these reasons, applicant submits that claim 122 is patentable over the cited combination.

i. Claim 123

Claim 123 should be considered separately and apart from the other claims. Claim 123 cannot be grouped with claims 114-122 and claim 128 because, unlike these claims, claim 123 requires the important distinctive feature of a specific tag syntax.

With regard to claim 123, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "templates" as disclosing components denoted on document templates using tag syntax, whereby the tag name identifies the component kind. However, WebWriter I on page 9, left column, discloses that output areas with scripts are denoted using a tag of the form

<output script="…">formatting string</output>

The tag name of this syntax is "output" while it is the script attribute that identifies what the script is to be called. In contrast, however, claim 123 explicitly requires the tag name to identify the component kind. In the office action dated July 1, 2009, on page34, the examiner writes "examiner interprets the tag name …". Applicant submits that the tag name is a well defined term in web technology and cannot be interpreted differently.

Therefore applicant submits that claim 123 is patentable over the cited combination.

j. Claim 124

Claim 124 should be considered separately and apart from the other claims. Claim 124 cannot be grouped with claims 114-123 and claim 128, because unlike these claims, claim 124 requires the important distinctive feature of instructions for displaying the component during editing and during normal use.

With regard to claim 124, the examiner cited WebWriter I at pages 2-9. Since the examiner did not recite a specific portion of the prior art, he did not establish a *prima facie* case of obviousness. The examiner refers to use of the term "browser" as disclosing second program instructions to generate browser code for displaying the component during editing. According to the base claim 114, second program instructions are part of selected components. As reasoned within sections B.ii, A.ii and C.i, WebWriter's scripts associated with dynamic areas are not executed during editing. The WebWriter Editor just displays a placeholder for scripts of dynamic areas during editing. In contrast, claim 124 requires second instructions in the components for displaying the components during editing. Therefore, applicant submits that claim 124 is patentable over the cited combination.

x. Claims 126-127

Claims 126-127 depend from Claim 125, and for all the same reasons are patentable over the cited combination.

a. Claim 126

Claim 126 and dependent Claim 127 should be considered as a group of claims. No other claims can be included in this group because claim 126 is the only method claim requiring a repetition of the running and the displaying step.

Claim 126 is a dependent method claim which requires the method of editing an application repeating the running and displaying steps. With regard to claim 126, the examiner cited WebWriter I at pages 2-9 as disclosing repeating the running and the displaying steps after applying a modification function. In applicants reading, as reasoned in sections A.i, A.ii, and C.i, WebWriter runs the application only after editing is finished and the document is saved. So there is no running step during editing and consequently also no repeating of the running step in WebWriter. In contrast, the claim requires repeating the running and the displaying steps after

applying a modification function. Therefore, applicant submits that claim 126 is patentable over the cited combination.

In addition, the examiner did not recite a specific portion of the prior art and so did not establish a *prima facie* case of obviousness.

b. Claim 127

Claims 127 depends on Claim 126, and for all the same reasons is patentable over the cited combination.

Respectfully submitted,

Date: May 12, 2010 By: /Richard A. Nebb/
Richard A. Nebb
Reg. No. 33,540

VIERRA MAGEN MARCUS & DENIRO LLP
575 Market Street, Suite 2500
San Francisco, California 94105
Telephone: 415.369.9660
Facsimile: 415.369.9665
Email: rnebb@vierramagen.com

CLAIMS APPENDIX

Complete Listing of Claims

1. A computer-readable medium encoded with computer programs having executable instructions for editing software applications that run on a data network which couples a server computer and a client computer, wherein the client computer runs a browser program, and whereupon request by the browser program, at least one of the applications generates generated documents for display by the browser program on a display device and responds to the request with the generated documents, comprising:

a document generator program running at least part of one of the applications being edited and generating the generated documents, said generated documents including additional editing features for interpretation by the browser program; and

an editor program dynamically operating on the generated documents displayed by the browser program via the editing features.

2. A computer-readable medium as in claim 1, further encoded with a plurality of components, and wherein the software applications comprise at least one document template capable of containing components, and wherein the editor provides features to insert, modify and delete a component on at least one document template, and wherein the document generator executes selected components on document templates.

3. A computer-readable medium as in claim 2, wherein at least one of the components reacts interactively on user input by executing instructions of said component on the server computer.

4. A computer-readable medium as in claim 3, wherein at least one of the components contains at least one other component.

5. A computer-readable medium as in claim 3, wherein the set of components on documents generated from at least one document template can vary for different requests of said document template.

6. A software development system having a data network which couples a server computer to a client computer, wherein the client computer includes a first software program for generating a request for one or more pages from the server computer and for displaying pages on a display device, and wherein the server computer includes a second software program for receiving and processing the request from the client computer, for generating and storing pages, and for transmitting pages to the client computer in response to requests, the server computer further comprising:

a data store,

a plurality of components residing in the data store, including at least one selected component executing first instructions contained in said selected component on the server computer;

a plurality of page templates residing in the data store, at least one page template having at least one selected component incorporated therein;

a server processor controlled by a third software program, said program providing instructions for selecting the page template based on the request from the client computer and instructions for generating a generated page from the page template for transmission to the client computer thereby calling first instructions; and

a component editor controlled by a fourth software program, said fourth software program providing instructions for editing selected components on the page template and instructions for making the first software program display a page for editing whereon the component has a similar appearance as on the generated page with the addition of editing features.

8. The development system of claim 6, wherein a component is nested within a component.

22. A computer-readable medium encoded with computer programs having executable instructions to edit and maintain applications using a web browser, comprising:

an editor program operating within the web browser on generated documents and having instructions for inserting, deleting, and modifying components on document templates; and

a document generator program having instructions for processing document templates, for executing said components, and for generating the generated documents from the document templates that are understandable by the web browser.

23. A computer readable medium as in claim 22, wherein the editor program operates a functional application in an edit mode permitting editing of said application directly in the web browser.

24. A computer readable medium as in claim 23 wherein at least one of the components contains instructions and can react on subsequent document requests containing user responses by executing selected instructions of said component.

25. A computer readable medium as in claim 24 encoded with:

component classes, each component class implementing one component kind; and

a parser program able to detect components marked on document templates;

wherein the document generator program works upon a document request using component classes to generate browser code; and

wherein the editor program is capable of showing a menu of components for insertion into the document templates.

26. A system having a data network which couples a server computer to a client computer, the server computer running an application to modify dynamic documents on the server computer, the server computer comprising:

a document store;

a first software program including instructions for transforming at least one first document retrieved from the document store into a second document having features which permit editing of the first document such that at least a part of the second document appears and functions similar to the run-time view of the first document; and

a second software program including instructions to receive information from the client computer and instructions to modify the first document stored in the document store.

27. The system of claim 26, wherein the first document includes at least one component being executed by the first software program, the component generating code for the run-time view of the first document.

28. The system of claim 27, wherein the second document includes handles and choosing one of the handles selects a component for an editing operation.

29. The system of claim 28, wherein at least one handle indicates the position of at least one component contained in the first document and said editing operation includes modifying the component, deleting the component, and displaying information regarding the component.

30. The system of claim 26, wherein the features include scripts.

31. The system of claim 30, wherein the scripts encapsulate information from the first document.

32. The system as in claim 26, wherein the features incorporate information regarding the first document into the second document.

33. A system as in claim 32, wherein the information incorporated into the second document is used on the client computer in order to send change requests for the first document to the server computer.

41. A computer-readable medium as in claim 1, the editor program comprising a client part for execution on the client computer.

42. A computer-readable medium as in claim 41, wherein the client part comprises instructions for execution during editing that are automatically downloaded from the server computer in a request prior to editing.

43. A computer-readable medium as in claim 26, additionally comprising at least one script for automatic download to the client that works in cooperation with the second document to permit editing of the first document.

51. A system having at least one computer running a second software program for editing components on web document templates for use with a first software program including first instructions for generating a document request to obtain at least one generated document from the second software program and for displaying the generated document, the second software program capable of receiving and processing the document request and of transmitting first documents to the first software program in response to requests, said system comprising:

a plurality of components containing instructions to generate browser code and each component having features to cooperate in editing the component,

a plurality of document templates,

the second software program transmitting, while processing selected requests, second documents to the first software program that make the first software program display a user interface for editing functions used for maintaining components on document templates, and

a third software program used by the second software program while processing selected document requests, the third software program including third instructions for modifying document templates in order to perform said editing functions.

52. The system of claim 51, wherein at least some components include fourth program instructions including steps to generate browser code for transmission to the first software program in response to a request from the first software program, wherein the browser code can differ for multiple requests for the same document template.

53. A system as in claim 52 having a data network, coupling the computer and a client computer, the first program running on the client computer.

54. A system as in claim 52 wherein second documents include HTML pages with embedded scripts and wherein the features include program instructions for collecting and

passing information to the editor and instructions for displaying additional editing features during editing.

55. The system of claim 52, wherein the editing functions include adding a component to a document template, removing a component from a document template, and modifying a component on a document template.

56. The system of claim 52, further comprising a fifth software program used by the second software program while processing selected document requests, the fifth software program including fifth instructions for generating generated documents from document templates thereby calling fourth program instructions.

57. The system of claim 56, wherein the generated documents include, if requested in edit mode, edit features for interpretation by the first software program.

58. The system of claim 56 further comprising instructions to allow the user to click on the generated documents to select items to perform edit functions on.

59. A software development system having at least one computer running an application for developing dynamic web documents, said dynamic web documents operating by being transformed into an end user's view upon a request by a web browser, the end user's view being provided to the browser for display on a display device in response to the request, comprising:

an editor program having instructions for dynamically editing dynamic web documents,

a document generator program having instructions for generating generated documents from dynamic web documents which look and function similar to the end user's view of the documents with the addition of editing features,

the editor program comprising first instructions for requesting that the document generator program processes a dynamic web document during editing thereby resulting in a generated document,

the system comprising second instructions for displaying at least some information items contained on said generated document in a view which allows the user to select an item to which a modification function will be applied,

the editor program comprising third instructions to modify the dynamic web document to perform said modification function.

60. The software development system of Claim 59 comprising a data network which couples a server computer and a client computer, the document generator program running on the server computer, the editor program at least partly running on the client computer.

61. The software development system of claim 60 comprising fourth instructions for execution during the document generation to collect edit-information for use by the editor program.

62. The software development system of claim 60, wherein the editor program uses a web browser for displaying said view.

63. The software development system of claim 60, comprising instructions for automatically repeating the request that the document generator processes the dynamic web document when required.

64. The software development system of Claim 59 further comprising a plurality of components including at least one component marked on said dynamic web document and including instructions for use by the document generator program to generate browser code.

65. The software development system of claim 64, wherein the editor program uses a web browser for displaying said view.

66. The software development system of claim 64, wherein the modification function includes insertion of a component, deletion of a component, and modification of a component.

67. The software development system of claim 59, wherein said view looks, except for editing features, similar to the end-user view of the generated document.

68. The software development system of claim 59 comprising sixth instructions to collect edit-information for use by the editor program, said sixth instructions for execution during the document generation.

69. The software development system of claim 68, wherein the editor program uses the edit-information to correctly modify the dynamic web document.

70. The software development system of claim 69, further comprising a plurality of components wherein the edit-information comprises position information on selected components marked on the dynamic web document.

71. The software development system of claim 59, wherein the editor program uses a web browser for displaying said view.

72. The software development system of claim 71, wherein the first instructions comprise seventh instructions for initiating a reload in the browser.

73. The software development system of claim 59 wherein the editor program comprises eighth instructions to display information on at least one element of at least one dynamic web document, that is replaced during document generation, without requesting that the document generator program generates a document.

74. A system having at least one computer running an application for developing document templates that are intended for transformation into generated documents for display by a first software program, the first software program including first instructions for generating a document request to obtain at least one generated document and for displaying the generated document on a display device, comprising:

a plurality of components having instructions to generate browser code for transmission to the first software program,

an editor program having instructions for performing editing functions to maintain components on document templates, the components having the ability to cooperate with the editor,

a plurality of document templates having said components denoted thereon, and

a document generator program having instructions to, upon document requests, generate generated documents from at least one document template for display by the first software program wherein the set of components on the generated documents can vary for different document requests for said document template.

75. The system as in claim 74, wherein the editing functions comprise adding a component, modifying a component, and deleting a component.

76. The system as in claim 74, wherein tag syntax is used to denote at least one component on at least one document template, whereby the tag name identifies the component kind.

77. The system of claim 74 comprising a server computer coupled to a client computer by a data network, the document generator program running on the server computer, and the editor program running, at least partly, on the client computer.

78. The system as in claim 74, wherein at least one component that can react interactively on subsequent document requests can be excluded from said generated document upon selected document requests for said document template.

79. The system as in claim 78 comprising third instructions to prevent excluded components from reacting on subsequent document requests.

80. A system as in claim 79, said third instructions comprising fourth instructions to, upon a first document request, store information in session memory on some of the components

that are present on the document generated based on the first document request, and fifth instructions to, upon subsequent document requests, only react on components that have been remembered in session memory thereby avoiding tampering with excluded components on the side of the first program.

81. A system as in claim 74 wherein at least one first component contains sixth instructions to decide upon a request for said document template about exclusion of components nested inside the first component from the generated document.

82. A system as in claim 74 the system providing an editable view taking the varying set of components into account.

83. A system as in claim 74 providing an editable view that includes and excludes selected components on different requests for said document template similar to the end user's view of the document template.

84. A system as in claim 74 wherein a document generated for at least one document template contains more components than the document template for at least one document request.

85. The system as in claim 74, wherein multiple instances of at least one third component denoted on the document template can be included in at least one of the documents generated from said document template.

86. The system as in claim 74, comprising seventh instructions to assign a unique identifier to each component instance of at least one seventh component, whereby the seventh component includes eighth instructions to qualify names generated into the browser code with the unique identifier.

87. A system as in claim 74, wherein at least one fourth component contains ninth instructions to decide upon a request about how many instances of components nested inside the fourth component are included in the documents generated from said document template.

88. A system as in claim 74 the editor program able to provide an editable view that includes multiple instances of selected components similar to the end user's view of the document template.

89. A system as in claim 74 wherein at least one sixth component includes tenth instructions to display the sixth component, the tenth instructions being used to generate browser code for displaying the sixth component during editing as well as during normal use of the component.

90. A system having at least one computer running an editor program for use with a web browser, the editor program having instructions for allowing the user to dynamically edit at least one document displayed by the browser on a display device, wherein scripts contained in said document remain running during editing, the editor program including a first software program for execution within the browser having instructions for processing selected clicks on the view of said document displayed in the browser by initiating editing functions.

91. The system as in claim 90 wherein the editor program includes instructions to display at least two windows, a first browser window displaying said document and a second window for displaying information on an element contained in said document.

92. The system as in claim 90 comprising a second software program having instructions for storing modifications on said document in cooperation with the first software program.

93. The system as in claim 92 further comprising a third program having instructions for transforming an original document into the document, the browser displaying the document as

said view looking similar to the original document and interpreting editing features contained in the document.

94. The system as in claim 93 wherein said original document is a dynamic document having components denoted thereon, the third software program comprising instructions for generating browser code in cooperation with selected instructions contained in the components.

95. The system as in claim 94 comprising a client computer connected to a server computer via a data network, wherein the browser together with the first software program is running on the client computer , and the second and the third software program run on the server computer.

96. The system as in claim 90 wherein links contained in said document stay functional allowing the user to browse and edit at the same time.

114. A system for displaying dynamically generated documents, the system having a data network coupling a server computer to a client computer, wherein the client computer has a first software program including first program instructions for generating a request to obtain at least one generated document from the server computer and for displaying the generated document on a display device, comprising:

a plurality of components having instructions for execution on the server computer, at least one of the components including first features adapted to cooperate with an editor in editing said component and second program instructions to generate browser code, and

a program having instructions on the server for dynamically generating generated documents for transfer to the client computer based on the data contained in a request initiated by the client computer, thereby using second program instructions of selected components.

115. The system of claim 128 wherein first features include fourth program instructions for passing information to the editor.

116. The system of claim 115 wherein at least part of said information is collected during execution of selected components on the server computer.

117. The system of claim 115 wherein said information is transmitted from the server computer to the client computer.

118. The system of claim 115 wherein said information includes attribute values of said component.

119. The system of claim 128 wherein first features include fifth instructions that display additional editing features of the component during editing.

120. The system of claim 119 wherein said editing features include handles.

121. The system of claim 128 wherein first features include an extension for use by the editor, said extension for enabling editing of an attribute value of the components .

122. The system of claim 121 wherein said extension enables display of a page for editing the attribute values of the components.

123. The system of claim 128 wherein at least one component is denoted on at least one document template using tag syntax, whereby the tag name identifies a component kind.

124. The system of claim 114 containing at least one component wherein second program instructions are used to generate browser code for displaying the component during editing and during normal use.

125. A method for dynamically editing an application that is built using components and that operates by generating documents comprising the steps of:

running at least part of the application on a computer, thereby executing selected components and generating a generated document,

displaying a view of the generated document,
selecting a component by clicking on selected portions of said view,
identifying the selected component in the source code of the application, and
initiating a modification function modifying the source code of the application.

126. The method of claim 125 wherein the running step and the displaying step are repeated after initiating the modification function.

127. The method of claim 125 further comprising collecting edit information for use by the identifying step.

128. The system of claim 114 additionally comprising a plurality of document templates with components denoted thereon, whereby the browser code generated by the components can vary for different requests of the same document template.

EVIDENCE APPENDIX
-NONE-

RELATED PROCEEDINGS APPENDIX
-NONE-